# COURS 4740A

-

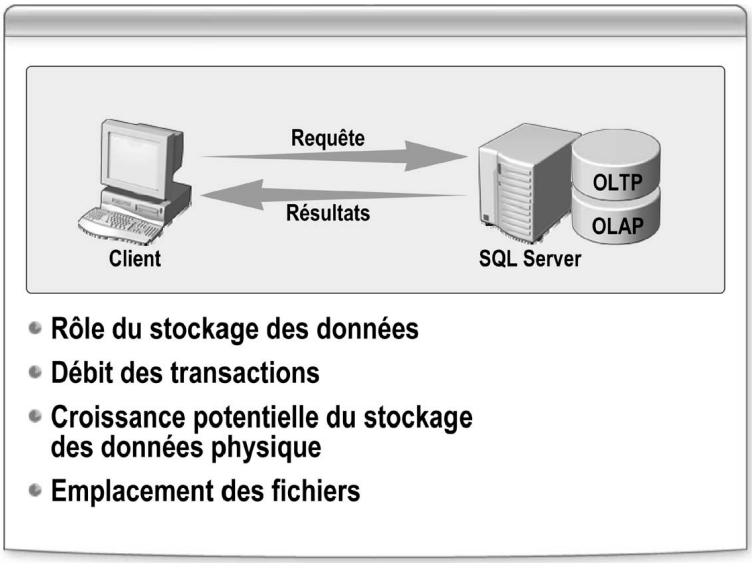
# IMPLEMENTATION D'UNE BASE DE DONNEES SQL SERVER 2005

# MODULE 1 : CREATION DE BASES DE DONNEES ET DE FICHIERS DE BASES DE DONNEES

## Leçon 1 : Création de bases de données

## 1. Eléments à prendre en considération pour la création de base de données

### 1.1Introduction




Vous pouvez utiliser SQL Server pour traiter des transactions, stocker et analyser des données, et créer des applications de base de données.
SQL Server est une famille de produits et de technologies qui répond aux impératifs de stockage de données des environnements OLTP (traitement transactionnel en ligne) et OLAP (traitement analytique en ligne).

SQL Server est un système de gestion de bases de données relationnelles (SGBDR) qui :

- gère le stockage des données pour les transactions et les analyses;
- stocke des données de nombreux types, notamment les données de type texte, numériques, XML (Extensible Markup Language), et des objets volumineux;
- répond aux demandes des applications clientes;
- utilise Transact-SQL, XML ou d'autres commandes SQL Server pour envoyer des demandes entre une application cliente et SQL Server.

Le composant SGBDR de SQL Server est chargé de:

- gérer les relations entre les données d'une base de données;
- vérifier que les données sont stockées correctement et que les règles qui définissent les relations au sein des données ne sont pas violées;
- récupérer toutes les données à un point de cohérence connue, dans le cas d'une défaillance du système.

### 1.2Base de données OLTP

Les tables relationnelles organisent les données dans une base de données OLTP pour réduire les informations redondantes et accroître la rapidité des mises à jour. SQL Server permet à un grand nombre d'utilisateurs d'exécuter des transactions et de modifier simultanément les données en temps réel dans les bases de données OLTP.

## 1.3Base de données OLAP

La technologie OLAP permet d'organiser et de synthétiser de grandes quantités de données afin qu'un analyste puisse évaluer des données rapidement et en temps réel. Microsoft SQL Server Analysis Services organise ces données pour prendre en charge un grand nombre de solutions professionnelles, de l'analyse et la génération de rapports d'entreprise à la modélisation des données, en passant par la prise de décision.

## 1.4Eléments à prendre en considération lors de la planification d'une base de données

Lorsque vous planifiez une nouvelle base de données, vous devez prendre plusieurs éléments en considération. La liste suivante présente certains de ces éléments.

- **Rôle du stockage des données.** Les bases de données OLTP et OLAP n'ont pas le même rôle ; par conséquent, leurs impératifs en termes de conception sont différents.
- **Débit des transactions.** Les impératifs des bases de données OLTP en termes de nombre de transactions pouvant être traitées par minute, heure ou jour sont élevés. Une conception efficace reposant sur un niveau de normalisation approprié, des index et des partitions de données peut contribuer à un débit de transactions élevé.
- **Croissance potentielle du stockage des données physique.** Pour de grandes quantités de données, le matériel pour la mémoire, l'espace disque et la puissance de l'unité centrale doivent être appropriés. L'estimation de la quantité de données
  que votre base de données contiendra durant les mois et les années à venir permettra de garantir que votre base de données continuera de fonctionner efficacement. Vous pouvez configurer vos bases de données de sorte que la taille des fichiers augmente automatiquement jusqu'à une limite maximale spécifiée. Toutefois, l'augmentation automatique de la taille des fichiers peut contribuer à diminuer les performances. Dans la plupart des solutions de base de données serveur, vous devez créer la base de données avec des fichiers de taille appropriée, contrôler l'utilisation de l'espace disque et réallouer plus d'espace uniquement lorsque cela s'avère nécessaire.
- **Emplacement des fichiers.** L'emplacement que vous choisissez pour les fichiers de base de données peut avoir une incidence sur les performances. Si vous avez la possibilité d'utiliser plusieurs lecteurs de disques, vous pouvez répartir vos fichiers de base de données sur plusieurs disques. Ainsi, SQL Server dispose de plusieurs connexions et de plusieurs en-têtes de disque pour une lecture et une écriture efficaces des données.

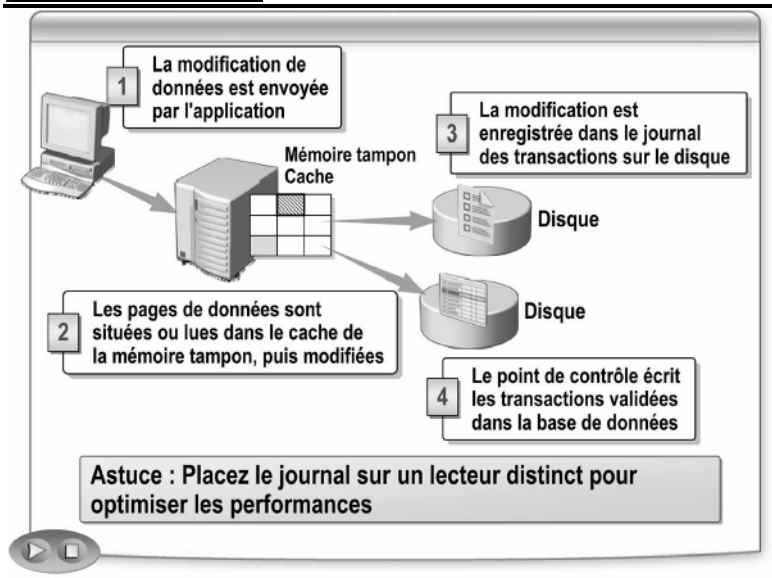
## 1.5Exemple de création de base de données

Vous pouvez créer une base de données à l'aide des outils graphiques de SQL Server Management Studio ou de l'instruction Transact-SQL CREATE DATABASE.

L'exemple suivant indique comment créer une base de données à l'aide de Transact-SQL.

```
CREATE DATABASE TestDB ON (NAME =
'TestDB_Data',
FILENAME = 'D:\DATA\TransactTestDB.mdf',
SIZE = 20 MB, FILEGROWTH = 0)
LOG ON (NAME = 'TestDB_Log',
FILENAME = 'D:\DATA\TestDB_Log.ldf',
SIZE = 5 MB,
FILEGROWTH = 0)
```

# 2. Enregistrement des transactions




## 2.1. Transaction

Une transaction correspond à une unité de travail dans une base de données. L'acronyme ACID est souvent utilisé pour décrire les caractéristiques d'une transaction. ACID signifie:

- **Atomicité.** Une transaction s'exécute de façon atomique: toutes les opérations définies dans la transaction sont entièrement accomplies ou pas du tout.
- **Cohérence.** Une transaction préserve systématiquement la cohérence des données.
- **Isolation.** Une transaction s'exécute en étant isolée de toute autre activité de la base de données ; les autres activités simultanées n'ont aucune incidence sur la transaction.
- **Durabilité.** Lorsqu'une transaction est validée, le système garantit que ses mises à jour perdureront, même si l'ordinateur tombe en panne après la validation.

SQL Server prend en charge les transactions implicites pour les instructions individuelles qui modifient les données, et les transactions explicites pour plusieurs instructions qui doivent être exécutées en tant qu'unité.

## 2.2. Journal des transactions

SQL Server enregistre chaque transaction dans un journal des transactions WAL (write-ahead log) pour préserver la cohérence de la base de données et contribuer à la récupération d'une défaillance de base de données. Le journal est une zone de stockage qui assure un suivi automatique des modifications apportées à une base de données.
SQL Server enregistre les modifications de disque dans le journal au fur et à mesure que les modifications sont exécutées et avant qu'elles ne soient écrites dans la base de données.

## 2.3. Processus d'enregistrement

Le processus d'enregistrement se déroule comme suit:

1. L'application envoie une modification de données.
2. Lorsqu'une modification est exécutée, les pages de données affectées sont chargées à partir du disque dans le cache de la mémoire tampon, à condition que les pages ne figurent pas déjà dans le cache de la mémoire tampon d'une requête antérieure.
3. Le journal enregistre chaque instruction de modification de données lorsque la modification intervient. La modification est toujours enregistrée dans le journal et écrite sur le disque avant que cette modification ne soit apportée dans la base de données. Ce type de journal est appelé journal WAL (write-ahead log).
4. Régulièrement, le processus de point de contrôle écrit toutes les transactions terminées dans la base de données sur le disque.

En cas de défaillance du système, le processus de récupération automatique utilise le journal des transactions pour restaurer par progression toutes les transactions validées et restaurer toutes les transactions non terminées.

Le journal utilise des marqueurs de transaction pendant la récupération automatique pour déterminer le point de départ et le point de terminaison d'une transaction.
Une transaction est considérée comme terminée lorsque le marqueur BEGIN TRANSACTION est associé à un marqueur COMMIT TRANSACTION. Les pages de données sont écrites sur disque lorsqu'un point de contrôle intervient.

### 2.4. Elements à prendre en compte pour l'emplacement des fichiers journaux

Pour améliorer les performances, il est recommandé de placer un fichier journal des transactions sur un disque physique différent de celui sur lequel résident vos fichiers de données. Cette séparation réduit la contention et permet à un jeu d'en-têtes de lecteur d'enregistrer les transactions dans le journal des transactions pendant que d'autres entêtes lisent les données à partir des fichiers de données. La mise à jour des données est rapide car les transactions peuvent être écrites immédiatement sur le disque, sans attendre la fin de la lecture des données. Étant donné que les fichiers journaux sont écrits séquentiellement, si le journal est stocké sur un disque dédié, les en-têtes de disque restent à l'emplacement correct pour l'opération d'écriture suivante.

## 3. Options de base de données

**Définissez les options de base de données comme suit :**

- **À l'aide de SQL Server Management Studio**
- **À l'aide de l'instruction ALTER DATABASE**

| Catégorie d'option | Fonction |
|---|---|
| Automatique | Détermine les comportements automatiques tels que les statistiques, la réduction et la fermeture de la base de données |
| Disponibilité | Détermine si la base de données est en ligne, en lecture seule, ainsi que les personnes autorisées à s'y connecter |
| Curseur | Détermine le comportement et l'étendue du curseur |
| Récupération | Détermine le mode de récupération de la base de données |
| SQL | Détermine les options de compatibilité ANSI telles que les valeurs ANSI NULL et les déclencheurs récursifs |

### 3.1. Introduction

Une fois la base de données créée, vous pouvez définir les options de base de données à l'aide des outils graphiques de SQL Server Management Studio ou de l'instruction Transact-SQL ALTER DATABASE.

Vous pouvez configurer plusieurs options de base de données, mais ne pouvez les définir que pour une base de données à la fois. Pour intégrer les options dans toutes les nouvelles bases de données, vous devez modifier la base de données **model**.

### 3.2. Catégorie des options de base de données

Plus de 25 options de base de données sont regroupées dans différentes catégories d'options pour simplifier la gestion. Le tableau suivant répertorie certaines des options les plus courantes.

| Catégorie d'option de base de données | Option de base de données | Description |
|---|---|---|
| Automatique | AUTO_CREATE_STATISTICS | Crée automatiquement toutes les statistiques manquantes exigées par une requête pour l'optimisation. La valeur par défaut est ON. |
| | AUTO_UPDATE_STATISTICS | Met automatiquement à jour les statistiques obsolètes exigées par une requête pour l'optimisation. La valeur par défaut est ON. |
| | AUTO_CLOSE | Ferme automatiquement une base de données lorsque le dernier utilisateur la quitte si la valeur est définie à ON. La valeur par défaut est OFF pour toutes les versions de SQL Server 2005, sauf pour SQL Server 2005 Express. |

| Catégorie d'option de base de données | Option de base de données | Description |
|---|---|---|
| (*suite*) | AUTO_SHRINK | Si cette option a la valeur ON, les fichiers de base de données peuvent faire l'objet d'une réduction périodique. La valeur par défaut est OFF. |
| Disponibilité | OFFLINE \| ONLINE \| EMERGENCY | Détermine si la base de données est en ligne ou hors connexion. L'option EMERGENCY interdit aux administrateurs qui ne sont pas des administrateurs système de se connecter, et elle configure la base de données en lecture seule. La valeur par défaut est ONLINE. |
| | READ_ONLY \| READ_WRITE | Détermine si les utilisateurs peuvent modifier les données. La valeur par défaut est READ_WRITE. |
| | SINGLE_USER \| RESTRICTED_USER \| MULTI_USER | Détermine les utilisateurs susceptibles de se connecter à la base de données. L'option SINGLE_USER permet à un seul utilisateur de se connecter. L'option RESTRICTED_USER permet aux membres du rôle de base de données **db_owner** et **dbcreator**, ainsi qu'aux rôles du serveur **sysadmin**, de se connecter. L'option MULTI_USER permet à tout utilisateur bénéficiant des autorisations de sécurité appropriées de se connecter. La valeur par défaut est MULTI_USER. |
| Curseur | CURSOR_CLOSE_ON_COMMIT | Ferme automatiquement les curseurs ouverts lorsqu'une transaction est validée. La valeur par défaut est OFF et les curseurs restent ouverts. |
| | CURSOR_DEFAULT | L'option CURSOR_DEFAULT_LOCAL limite l'étendue du curseur. Cette option est locale au lot, à la procédure stockée ou au déclencheur dans lequel le curseur a été créé. CURSOR_DEFAULT_GLOBAL est le paramètre d'option par défaut ; l'étendue du curseur est globale à la connexion. |

| Catégorie d'option de base de données | Option de base de données | Description |
|---|---|---|
| Récupération | RECOVERY | L'option FULL assure une récupération complète à partir de la défaillance d'un support ; il s'agit de la valeur par défaut. L'option BULK_LOGGED utilise une plus petite quantité d'espace du journal car l'enregistrement est minime, mais le risque d'exposition est plus important. L'option SIMPLE récupère la base de données uniquement au point de la dernière sauvegarde complète de la base ou de la dernière sauvegarde différentielle. |
| | PAGE_VERIFY | Permet à SQL Server de détecter les opérations d'E/S non terminées à la suite d'une coupure de courant ou d'autres pannes du système. CHECKSUM stocke une valeur calculée dans l'en-tête de page en fonction du contenu de la page. Cette valeur est recalculée puis comparée à la version stockée lorsque les pages de données sont lues à partir du disque. Il s'agit de la valeur par défaut. L'option TORN_PAGE_DETECTION stocke un bit spécifique pour chaque secteur de 512 octets d'une page de données de 8 kilo-octets dans l'en-tête de la page. Les bits stockés dans l'en-tête de la page sont comparés aux informations réelles du secteur concerné lorsque les pages de données sont lues à partir du disque. |
| SQL | ANSI_NULL_DEFAULT | Permet à l'utilisateur de déterminer la valeur NULL par défaut de la base de données. La valeur par défaut est OFF dans SQL Server 2005, par conséquent NOT NULL. |
| | ANSI_NULLS | Si la valeur ON est spécifiée, toutes les comparaisons à une valeur NULL sont évaluées à UNKNOWN. Si la valeur OFF est spécifiée, les comparaisons de valeurs non UNICODE à une valeur NULL sont évaluées à TRUE si les deux valeurs sont NULL. Par défaut, l'option de base de données ANSI_NULLS est OFF. |
| (*suite*) | RECURSIVE_TRIGGERS | Détermine si le déclenchement récursif des déclencheurs AFTER est autorisé. La valeur par défaut est OFF, ce qui empêche la récursivité directe. |

# 4. Source d'informations de base de données

| Source d'informations | Description |
|---|---|
| SQL Server Management Studio | Outil visuel qui affiche les métadonnées des bases de données dans l'environnement de gestion |
| Affichages catalogue | Fournissent les métadonnées concernant les objets de base de données qui retournent des lignes d'informations |
| Fonctions de métadonnées | Retournent une seule valeur d'informations de métadonnées par fonction |
| Procédures stockées système | Récupèrent les métadonnées à l'aide de procédures stockées |

## 4.1. Introduction

Vous pouvez consulter les métadonnées système concernant vos bases de données de deux manières. Lorsque vous devez afficher des informations relatives à un objet de la base de données, la méthode la plus simple consiste à utiliser SQL Server Management Studio. Lorsque vous écrivez des applications qui récupèrent les métadonnées concernant les objets des bases de données, vous devez utiliser Transact-SQL pour interroger les affichages catalogue fournis par le système, utiliser les fonctions système ou exécuter les procédures stockées système.

## 4.2. SQL Server Management Studio

SQL Server Management Studio fournit des outils graphiques pour afficher les métadonnées des bases de données dans l'environnement de gestion. Le tableau suivant répertorie les principaux outils utilisés.

| Outil de SQL Server Management Studio | Description |
|---|---|
| Explorateur d'objets | L'Explorateur d'objets est un outil graphique permettant de rechercher et de gérer les serveurs, les bases de données et les objets de base de données. |
| fenêtre Propriétés | Chaque objet de base de données dans l'Explorateur d'objets possède une fenêtre Propriétés associée à laquelle l'utilisateur peut accéder en cliquant avec le bouton droit sur l'objet puis en cliquant sur **Propriétés**. La fenêtre Propriétés varie selon le type d'objet sélectionné. |
| Rapports | SQL Server Management Studio inclut des rapports pour différents nœuds fournis dans l'Explorateur d'objets par le moteur Report Server de SQL Server. Les nœuds fréquemment utilisés qui peuvent afficher des rapports sont les suivants :<br>■ **Serveur**<br>■ **Base de données**<br>■ **Service Broker**, sous un nœud **Base de données**<br>■ **Connexions**, sous le nœud **Sécurité**<br>■ **Gestion** |

## 4.3. Affichages catalogue

Les affichages catalogue vous permettent d'interroger les métadonnées liées aux objets de base de données SQL Server, notamment les tables, les procédures stockées et les contraintes. Quelques affichages catalogue fournissent des informations pour l'ensemble du serveur, mais la majorité des affichages sont spécifiques à une base de données.

Les affichages catalogue sont répertoriés dans le dossier Views\System de chaque base de données dans l'Explorateur d'objets de SQL Server Management Studio. Bien que vous puissiez interroger les affichages catalogue à l'aide de la syntaxe Transact-SQL standard comme pour les affichages définis par l'utilisateur, ces affichages ne sont pas réellement implémentés comme des affichages traditionnels sur les tables sous-jacentes. En fait, ils interrogent directement les métadonnées système. Il existe plus de 200 affichages catalogue, lesquels sont définis dans le schéma **sys**.

Les affichages catalogue sont organisés en catégories d'après leur fonction. Le tableau suivant présente plusieurs catégories clés et quelques affichages catalogue couramment utilisés dans chaque catégorie.

| Catégorie | Affichage catalogue | Description |
|---|---|---|
| Bases de données et fichiers | **sys.databases** | Retourne une ligne pour chaque base de données sur le serveur |
| | **sys.database_files** | Retourne une ligne pour chaque fichier d'une base de données |
| Objets | **sys.columns** | Retourne une ligne pour chaque colonne d'un objet contenant des colonnes (par exemple, une table ou une vue) |
| | **sys.events** | Retourne une ligne pour chaque événement pour lequel un déclencheur ou une notification sont activés |
| | **sys.indexes** | Retourne une ligne pour chaque index ou segment de mémoire d'un objet au format tabulaire |
| | **sys.tables** | Retourne une ligne pour chaque table de la base de données |
| | **sys.views** | Retourne une ligne pour chaque affichage dans la base de données |
| Schémas | **sys.schemas** | Retourne une ligne pour chaque schéma défini dans la base de données |
| Sécurité | **sys.database_permissions** | Retourne une ligne pour chaque autorisation définie dans la base de données |
| | **sys.database_principals** | Retourne une ligne pour chaque entité de sécurité de la base de données |
| | **sys.database_role_members** | Retourne une ligne pour chaque membre de chaque rôle de base de données |

### 4.4.Procédures stockées

SQL Server 2005 propose de nombreuses procédures stockées système permettant de récupérer les métadonnées des bases de données. Ces procédures constituent une autre méthode pour obtenir les informations fournies par les affichages catalogue, et certaines acceptent des arguments de procédure afin d'autoriser la personnalisation des ensembles de résultats.

La liste suivante décrit quelques-unes des procédures stockées système les plus courantes parmi les centaines de procédures disponibles:

■ **sp_databases.** Répertorie les bases de données qui sont disponibles dans une instance de SQL Server ou qui sont accessibles par le biais d'une passerelle de base de données.

■ **sp_stored_procedures.** Retourne la liste des procédures stockées dans la base de données active.

■ **sp_help.** Fournit des informations sur un objet de base de données, un type de données défini par l'utilisateur ou un type de données fourni par SQL
Server 2005.

# 5. Application pratique : Création de base de données

## 5.1.Création de base de données à l'aide de SQL Server Management studio

(Les différentes références à MIAMI correspond au nom de votre serveur)

Pour créer une base de données à l'aide de SQL Server Management Studio, procédez comme suit:

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, sur **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, cliquez sur **Se connecter**.
3. Dans l'Explorateur d'objets, cliquez avec le bouton droit sur **Bases de données**, puis cliquez sur **Nouvelle base de données**.
4. Dans la boîte de dialogue **Nouvelle base de données**, entrez les informations figurant dans le tableau suivant.

| Propriété | Valeur |
|---|---|
| **Nom de la base de données** | **TestDB** |
| **Taille initiale** de **TestDB** | **20** |

5. Dans la colonne **Croissance automatique** de l'entrée **TestDB**, cliquez sur le bouton(...).
6. Dans la boîte de dialogue **Modifier la croissance automatique** pour **TestDB**, désactivez la case à cocher **Activer la croissance automatique**, puis cliquez sur **OK**.
7. Pour le paramètre **Taille initiale** de l'entrée **TestDB_log**, utilisez la valeur **5 Mo**.
8. Dans la colonne **Croissance automatique** de l'entrée **TestDB_log**, cliquez sur le bouton (...).
9. Dans la boîte de dialogue **Modifier la croissance automatique** pour **TestDB_log**, désactivez la case à cocher **Activer la croissance automatique**, puis cliquez sur **OK**.
10. Dans la boîte de dialogue **Nouvelle base de données**, cliquez sur **OK** pour créer la base de données.
11. Dans l'Explorateur d'objets, développez le dossier **Bases de données** pour vérifier que la base **TestDB** a été créée; si **TestDB** n'est pas répertoriée, cliquez avec le bouton droit sur le dossier **Bases de données**, puis cliquez sur **Actualiser**.
12. Ne fermez pas SQL Server Management Studio, car vous l'utiliserez dans la procédure suivante.

## 5.2. Création de base de données à l'aide de Transact-SQL

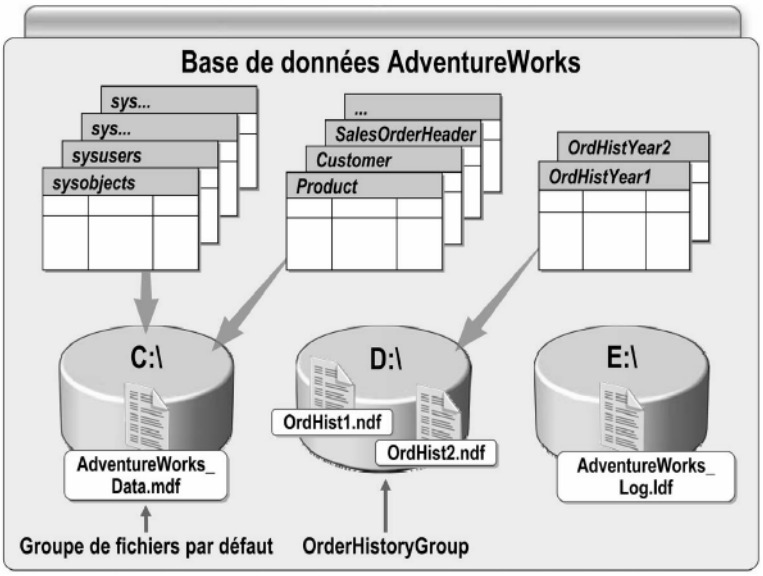
1. Pour créer une base de données à l'aide de Transact-SQL, procédez comme suit : Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
2. Dans la nouvelle fenêtre de requête vierge, tapez le code Transact-SQL suivant (chaque paramètre FILENAME doit figurer sur une seule ligne).

```
CREATE DATABASE TransactTestDB
ON (NAME = 'TransactTestDB',
     FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\TransactTestDB.mdf',
     SIZE = 20 MB,
     FILEGROWTH = 0)
LOG ON (NAME = 'TransactTestDB_Log',
     FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\TransactTestDB.ldf',
     SIZE = 5 MB,
     FILEGROWTH = 0)
```

3. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
4. Une fois la commande exécutée, cliquez avec le bouton droit sur le dossier **Bases de données** dans l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que la base de données **TransactTestDB** a été créée.
5. Fermez SQL Server Management Studio. Cliquez sur **Non** lorsque vous êtes invité à enregistrer les fichiers.

# Leçon 2 : Création de groupes de fichiers

## 1. Qu'est-ce que les groupes de fichiers ?




### 1.1Introduction

Un groupe de fichiers est une collection logique de fichiers de données qui permet aux administrateurs de gérer tous les fichiers du groupe de fichiers comme un élément seul. La possibilité de contrôler l'emplacement physique des différents objets de la base de données peut offrir plusieurs avantages en termes de facilité de gestion et de performances. Par exemple, vous pouvez utiliser plusieurs groupes de fichiers pour déterminer comment les données d'une base de données sont stockées physiquement sur les périphériques de stockage et pour séparer les données en lecture/écriture des données en lecture seule

### 1.2Types de groupes de fichiers

SQL Server 2005 possède un groupe de fichiers primaire et peut également inclure des groupes de fichiers définis par l'utilisateur.

■ Le groupe de fichiers primaire contient le fichier de données primaire avec les tables système. Le fichier des données primaire utilise généralement l'extension .mdf.

■ Un groupe de fichiers défini par l'utilisateur est composé de fichiers de données qui sont regroupés pour l'allocation et pour des raisons administratives. Ces autres fichiers de données sont appelés fichiers de données secondaires, et ils utilisent généralement l'extension .ndf.

### 1.3Exemple de scénario pour plusieurs groupes de fichiers

L'illustration propose un exemple de placement des fichiers de base de données sur des disques distincts, comme indiqué dans la liste suivante:

■ Vous pouvez créer des groupes de fichiers définis par l'utilisateur pour séparer les fichiers qui font l'objet de nombreuses interrogations de ceux qui sont fortement modifiés. Dans l'illustration, les fichiers OrdHist1.ndf et OrdHist2.ndf sont placés sur un disque distinct de celui où résident les tables **Product**, **Customer** et **SalesOrderHeader**, car ils sont interrogés pour l'aide à la prise de décision ; ils ne sont pas mis à jour avec les informations de commande actuelles.

■ Vous pouvez également placer les fichiers OrdHist1.ndf et OrdHist2.ndf sur des disques distincts s'ils sont tous deux lourdement interrogés.

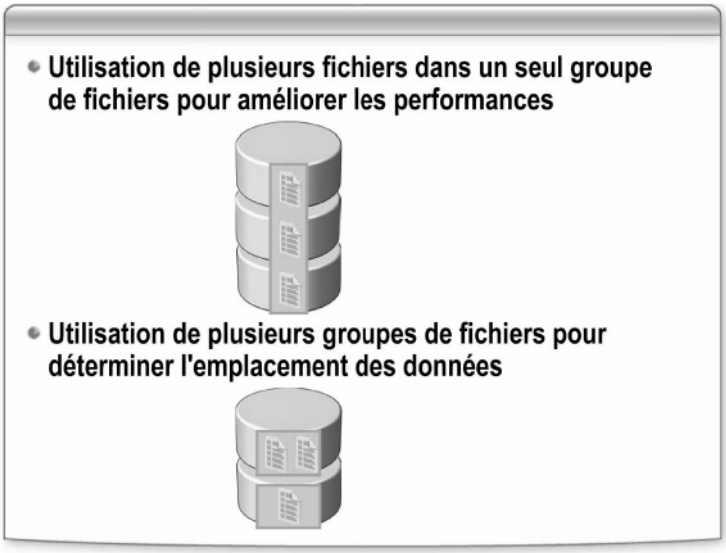
■ Vous ne pouvez pas placer les fichiers du journal des transactions dans des groupes de fichiers. L'espace réservé aux journaux des transactions est géré indépendamment de l'espace réservé aux données. Les journaux des transactions utilisent généralement l'extension .ldf.

L'exemple de code Transact-SQL suivant utilise l'instruction CREATE DATABASE pour mettre en œuvre ce scénario.

```
CREATE DATABASE [AdventureWorks] ON PRIMARY
( NAME = 'AdventureWorks_Data', FILENAME = 'C:\AdventureWorks_Data.mdf' ), FILEGROUP
[OrderHistoryGroup]
( NAME = 'OrdHist1', FILENAME = 'D:\OrdHist1.ndf' ),
( NAME = 'OrdHist2', FILENAME = 'D:\OrdHist2.ndf' )
LOG ON
( NAME = 'AdventureWorks_log', FILENAME = 'E:\AdventureWorks_log.ldf')
```

Vous pouvez également utiliser l'instruction ALTER DATABASE pour ajouter ou supprimer des fichiers et des groupes de fichiers dans les bases de données existantes.

# 2. Quand créer des groupes de fichiers ?




## 2.1. Introduction

Vous pouvez créer plusieurs fichiers de données sur des disques indépendants et créer un groupe de fichiers défini par l'utilisateur pour qu'il contienne les fichiers.
Les groupes de fichiers sont principalement utilisés pour deux raisons : l'amélioration des performances et le contrôle de l'emplacement physique de données.

## 2.2. Utilisation de plusieurs fichiers dans un groupe de fichiers unique pour améliorer les performances

Bien que le recours à la technologie RAID (Redundant Array of Independent Disks) soit recommandé pour améliorer les performances d'une base de données, vous pouvez affecter plusieurs fichiers sur des disques distincts à un seul groupe de fichiers pour optimiser les performances en implémentant des bandes de données dans SQL Server. Dans la mesure où SQL Server utilise une stratégie de remplissage proportionnel lors de l'écriture des données dans un groupe de fichiers, les données sont agrégées par bandes sur les fichiers, et par conséquent, sur les partitions de disques physiques. Cette approche permet un contrôle plus fin de l'agrégat par bandes de données que celui qui est obtenu en créant un jeu de volumes ou d'agrégats par bandes dans le système d'exploitation Windows ou en utilisant un contrôleur de baie de disques RAID.

## 2.3. Utilisation de plusieurs groupes de fichiers pour déterminer l'emplacement physique des données

Si vous souhaitez utiliser des groupes de fichiers pour simplifier la maintenance ou atteindre des objectifs de conception, vous pouvez:

■ stocker les données en lecture/écriture séparément des données en lecture seule pour que différents types d'activité d'E/S disque restent séparés;

■ stocker les index sur des disques distincts de ceux des tables, ce qui peut entraîner une amélioration des performances;

■ sauvegarder ou restaurer certains fichiers ou groupes de fichiers au lieu de sauvegarder ou de restaurer l'intégralité d'une base de données. Pour les bases de données volumineuses, la sauvegarde des fichiers ou des groupes de fichiers peut être nécessaire afin que la stratégie de sauvegarde et de restauration soit efficace;

■ regrouper les tables et les index répondant aux mêmes impératifs de maintenance dans les mêmes groupes de fichiers. Vous pouvez effectuer une maintenance sur certains objets plus fréquemment que sur d'autres. Par exemple, si vous créez deux groupes de fichiers et leur affectez des tables, vous pouvez exécuter des tâches de maintenance quotidiennes sur les tables dans un groupe quotidien et des tâches de maintenance hebdomadaires sur les tables dans un groupe hebdomadaire. Cette approche limite les conflits avec les disques entre les deux groupes de fichiers;

■ séparer les tables utilisateur et les autres objets de la base de données des tables système dans le groupe de fichiers primaire. Vous devez également modifier le groupe de fichiers par défaut pour empêcher qu'une augmentation inattendue de la taille des tables cantonne les tables système dans le groupe de fichiers primaire;

■ stocker les partitions d'une table partitionnée dans plusieurs groupes de fichiers. Cette méthode est utile pour séparer physiquement les données avec des besoins différents en termes d'accès dans une seule table. Elle peut également apporter des avantages en termes de facilité de gestion et de performances.

# 3. Application pratique : création de groupes de fichiers

## 3.1. Création d'un groupe de fichiers à l'aide de SQL Server Management Studio

Pour créer un groupe de fichiers à l'aide de SQL Server Management Studio, procédez comme suit:

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, sur **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Si l'Explorateur d'objets n'est pas visible, cliquez sur **Explorateur d'objets** dans le menu **Affichage**.
4. Dans l'Explorateur d'objets, développez **Bases de données**.
5. Cliquez avec le bouton droit sur **TestDB**, puis cliquez sur **Propriétés**.
6. Dans la boîte de dialogue **Propriétés de la base de données - TestDB**, cliquez sur **Groupes de fichiers** dans le volet **Sélectionner une page**, puis cliquez sur **Ajouter**.
7. Entrez les paramètres indiqués dans le tableau suivant. **Propriété**

| **Valeur** | |
|---|---|
| **Nom** | **SECONDARY** |
| **Par défaut** | Sélectionné |

8. Cliquez sur **Fichiers** dans le volet **Sélectionner une page**, puis cliquez sur **Ajouter**.

9. Entrez les paramètres indiqués dans le tableau suivant.

| Propriété | Valeur |
|---|---|
| **Nom logique** | **TestDB2** |
| **Type de fichier** | **Données** |
| **Groupe de fichiers** | **SECONDARY** |

10. Dans la boîte de dialogue **Propriétés de la base de données -TestDB**, cliquez sur **OK** pour créer le nouveau fichier et le groupe de fichiers.
11. Si le nœud **TestDB** n'est pas sélectionné, cliquez sur **TestDB**.
12. Sous l'onglet **Résumé**, cliquez sur **Rapport**. Lorsque le rapport est affiché, développez **Espace disque utilisé par les fichiers de données**. Le rapport doit afficher les deux groupes de fichiers et la liste des fichiers de chaque de groupe.
13. Ne fermez pas SQL Server Management Studio, car vous l'utiliserez dans la procédure suivante.

## 3.2. Création d'un groupe de fichiers à l'aide de Transact-SQL

Pour créer un groupe de fichiers à l'aide de Transact-SQL, procédez comme suit:

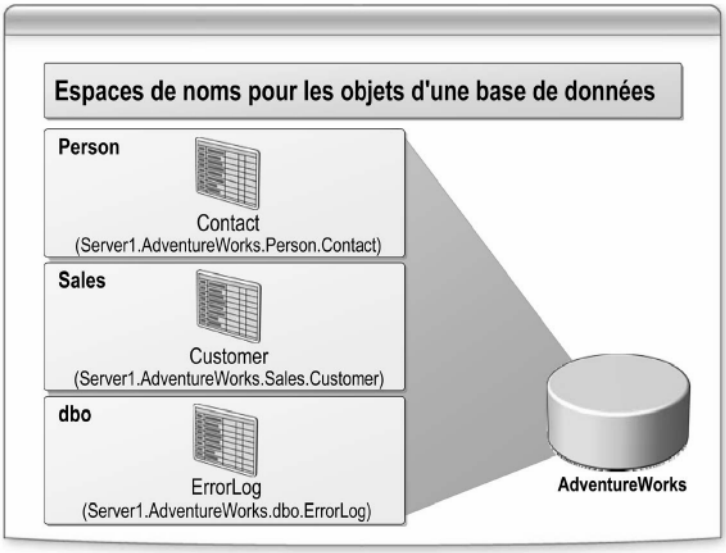
1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
2. Dans la nouvelle fenêtre de requête vierge, tapez le code Transact-SQL suivant.

```
ALTER DATABASE [TransactTestDB]
ADD FILEGROUP [SECONDARY]
GO
ALTER DATABASE [TransactTestDB]
ADD FILE (
      NAME = 'Test2',
      FILENAME ='C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\Test2.ndf')
TO FILEGROUP [SECONDARY]
GO
ALTER DATABASE [TransactTestDB] MODIFY FILEGROUP [SECONDARY] DEFAULT
GO
```

3. Cliquez sur le bouton **Exécuter** dans la barre d'outils. Si le nœud **TransactTestDB** n'est pas sélectionné, cliquez sur **TransactTestDB**.
4. Cliquez sur l'onglet **Résumé**, puis cliquez sur **Rapport**. Lorsque le rapport est affiché, développez **Espace disque utilisé par les fichiers de données**. Le rapport doit afficher les deux groupes de fichiers et la liste des fichiers de chaque de groupe.
5. Fermez SQL Server Management Studio. Cliquez sur **Non** lorsque vous êtes invité à enregistrer les fichiers.

# Leçon 3 : Création de schémas

## 1. Qu'est-ce que les schémas ?




### 1.1Introduction

Les objets d'une base de données (tels que les tables, les vues et les procédures stockées) sont créés dans un schéma. Vous devez impérativement savoir ce qu'est un schéma avant de planifier une base de données SQL Server 2005 et de l'implémenter.

### 1.2Schémas en tant qu'espace de noms

Un *schéma* est un espace de noms pour les objets d'une base de données. En d'autres termes, un schéma définit une limite dans laquelle tous les noms sont uniques. Compte tenu que les noms de schémas doivent être uniques dans la base de données, chaque objet d'une base de données possède un nom complet au format *serveur.base_de_données.schéma.objet*. Dans une base de données, vous pouvez le raccourcir au format *schéma.objet*.

L'illustration précédente présente trois schémas dans la base de données **AdventureWorks** dans une instance SQL Server nommée **Server1**. Les schémas sont nommés **Person**, **Sales** et **dbo**. Chacun de ces schémas contient une table, et le nom complet de la table inclut le nom du serveur, le nom de la base de données et le nom du schéma. Par exemple, le nom complet de la table **ErrorLog** dans le schéma **dbo** est **Server 1 .AdventureWorks.dbo.ErrorLog**.

Dans les versions antérieures de SQL Server, l'espace de noms d'un objet était déterminé par le nom d'utilisateur de son propriétaire. Dans SQL Server 2005, les schémas sont indépendants de la propriété des objets. Cette séparation offre les avantages suivants:

- Flexibilité accrue lorsque vous organisez les objets de base de données en espaces de noms, car le regroupement d'objets dans des schémas ne dépend pas de la propriété des objets.
- Gestion des autorisations simplifiée, car une autorisation peut être accordée à l'échelle du schéma aussi bien que sur des objets spécifiques.
- Gestion améliorée, car la suppression d'un utilisateur n'entraîne pas l'obligation de renommer tous les objets que l'utilisateur possède.

### 1.3Exemples des schémas

La base de données **AdventureWorks** utilise les schémas suivants pour organiser ses objets de base de données en espaces de noms:

- **HumanResources**
- **Person**
- **Production**
- **Purchasing**
- **Sales**

Par exemple, vous désignez la table **Employee** dans le schéma **HumanResources** sous la forme **HumanResources.Employee**.

### 1.4Schéma dbo

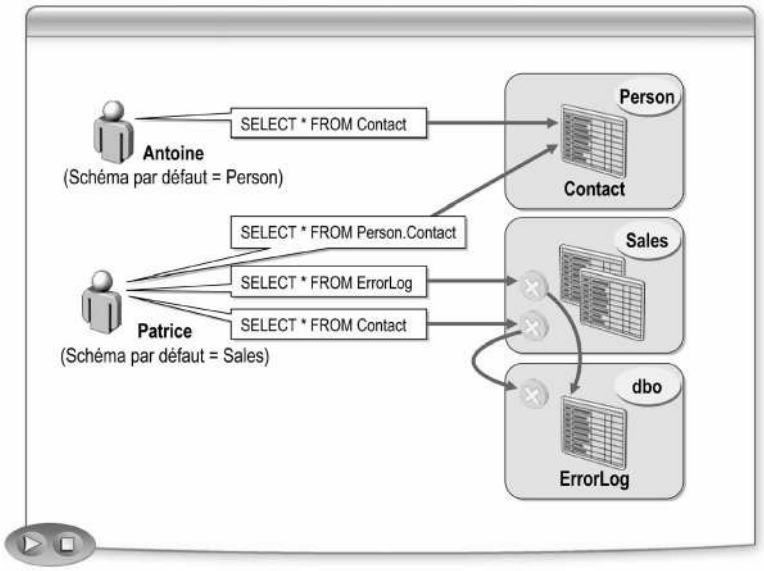
Chaque base de données contient un schéma nommé **dbo**. Le schéma **dbo** est le schéma par défaut de tous les utilisateurs qui ne possèdent pas d'autre schéma par défaut défini explicitement.

### 1.5Création d'un schéma

Pour créer un schéma, utilisez L'Explorateur d'objets de SQL Server Management Studio ou l'instruction CREATE SCHEMA, comme l'illustre l'exemple suivant.

```
Use AdventureWorks
GO
CREATE SCHEMA Sales
GO
```

## 2. Fonctionnement de la résolution de noms d'objets




## 2.1. Introduction

Lorsqu'une base de données contient plusieurs schémas, la résolution de noms d'objets peut se révéler délicate et complexe. Par exemple, une base de données peut contenir deux tables nommées **Order** dans deux schémas distincts, **Sales** et **dbo**. Les noms complets des objets de la base de données sont sans ambiguïté: **Sales.Order** et **dbo.Order**, respectivement. Toutefois, l'utilisation du nom non qualifié **Order** peut produire

des résultats inattendus. Vous pouvez affecter un schéma par défaut aux utilisateurs pour qu'ils puissent déterminer comment les noms d'objets non qualifiés sont résolus.

## 2.2. Fonctionnement de la résolution de noms

SQL Server 2005 utilise le processus suivant pour résoudre un nom d'objet non qualifié:

1. Si l'utilisateur possède un schéma par défaut, SQL Server tente de trouver l'objet dans le schéma par défaut.
2. Si l'objet est introuvable dans le schéma par défaut de l'utilisateur, ou si l'utilisateur ne possède aucun schéma par défaut, SQL Server tente de trouver l'objet dans le schéma **dbo**.

Par exemple, un utilisateur doté du schéma par défaut **Person** exécute l'instruction Transact-SQL suivante:

```
SELECT * FROM Contact
```

SQL Server 2005 tente tout d'abord de résoudre le nom d'objet en **Person.Contact**.

Si le schéma **Person** ne contient pas d'objet nommé **Contact**, SQL Server tente alors de résoudre le nom d'objet en **dbo.Contact**.

## 2.3. Affectation d'un schéma par défaut

Pour affecter un schéma par défaut à un utilisateur, utilisez la boîte de dialogue Propriétés de Utilisateur de la base de données, ou spécifiez le nom du schéma dans la clause DEFAULT_SCHEMA de l'instruction CREATE USER ou ALTER USER.

Par exemple, le code Transact-SQL suivant affecte **Sales** comme schéma par défaut de l'utilisateur **Patrice**: `ALTER USER Patrice WITH DEFAULT_SCHEMA = Sales`

# 3. Application pratique : création d'un schéma : création d'un schéma

## 3.1. Création d'un schéma à l'aide de SQL Server Management studio

Pour créer un schéma à l'aide de SQL Server Management Studio, procédez comme suit:

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, sur **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Si l'Explorateur d'objets n'est pas visible, cliquez sur **Explorateur d'objets** dans le menu **Affichage**.
4. Dans l'Explorateur d'objets, développez **Bases de données**, **TestDB**, puis **Sécurité**.
5. Cliquez avec le bouton droit sur **Schémas**, puis cliquez sur **Nouveau schéma**.
6. Dans la boîte de dialogue **Schéma - Nouveau**, entrez **Sales** pour **Nom du schéma**, puis cliquez sur **OK**.
7. Dans Explorateur d'objets, développez **Schémas**, puis confirmez que le schéma **Sales** existe.
8. Ne fermez pas SQL Server Management Studio, car vous l'utiliserez dans la procédure suivante.

## 3.2. Création d'un schéma à l'aide de Transact-SQL

Pour créer un schéma à l'aide de l'instruction Transact-SQL CREATE SCHEMA, procédez comme suit:

1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
2. Dans la nouvelle fenêtre de requête vierge, tapez le code Transact-SQL suivant.

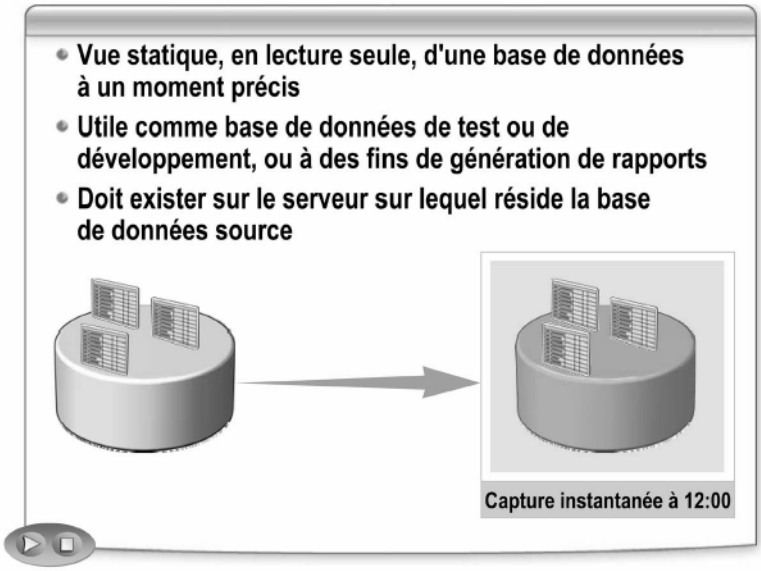
```
Use TransactTestDB
GO
CREATE SCHEMA [Marketing]
GO
```

3. Cliquez sur le bouton **Exécuter** dans la barre d'outils.

4. Dans l'Explorateur d'objets, développez **Bases de données**, **TransactTestDB**, **Sécurité** et **Schémas**, puis confirmez que le schéma **Marketing** existe.
5. Fermez SQL Server Management Studio. Cliquez sur **Non** lorsque vous êtes invité à enregistrer les fichiers.

# Leçon 4 : Création de captures instantanées de base de données

## 1. Qu'est-ce que les captures instantanées de base de données ?




### 1.1. Introduction

Dans de nombreux scénarios, une simple copie de la base de données, appelée *capture instantanée*, est utile comme base de données de secours semi-automatique, base de données de test et de développement, ou simplement comme base de données de rapports. Vous utilisez la clause AS SNAPSHOT OF de l'instruction CREATE DATABASE pour créer une capture instantanée de base de données.

### 1.2. Définition

Une capture instantanée est une vue statique en lecture seule d'une base de données à un moment spécifique, qui ne change pas après la création de la capture instantanée. La base de données à partir de laquelle la capture instantanée est créée est désignée sous le nom de base de données source.

Les captures instantanées de base de données peuvent être utiles en tant que point de restauration rapide en cas de dommages accidentels ou malveillants des données de la base de données. Toutefois, une capture instantanée de base de données ne peut pas remplacer une sauvegarde, car elle ne contient pas tous les enregistrements de la base de données.

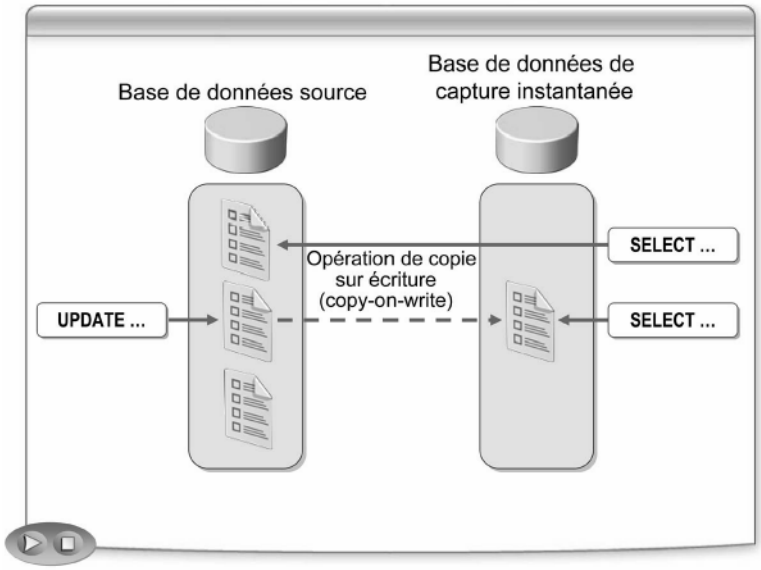
### 1.3. Restrictions liées à la création de captures instantanées

Il existe une limitation pour une capture instantanée de base de données : cette dernière doit être située sur le même serveur que la base de données source. Les restrictions suivantes s'appliquent également aux captures instantanées de base de données :

- Il est impossible de créer des captures instantanées de base de données pour les bases de données **model**, **master** ou **tempdb**.
- Les captures instantanées de base de données ne peuvent pas être sauvegardées ou restaurées.

- Les captures instantanées de base de données ne peuvent pas être attachées ou détachées.
- Les captures instantanées de base de données ne peuvent pas être créées sur des partitions FAT32 ou brutes.
- Toutes les captures instantanées créées sur une base de données doivent être supprimées avant que la base de données ne soit supprimée.
- SQL Server Management Studio ne prend pas en charge la création de captures instantanées. Par conséquent, les captures instantanées de base de données peuvent être créées uniquement à l'aide de Transact-SQL.

# 2. Fonctionnement des captures instantanées de base de données




## 2.1. Introduction

Les captures instantanées de base de données préservent leur vue statique d'une base de données source en stockant des copies des données de pré-modification lorsque les mises à jour interviennent dans la base de données source. Ces informations copiées sont ensuite retournées dans le cadre d'une requête normale.

## 2.2.Modification des données

SQL Server 2005 utilise la technologie de copie sur écriture (« copy-on-write ») pour implémenter des captures instantanées de base de données sans subir la surcharge liée à la création d'une copie complète de la base de données. Une capture instantanée de base de données est initialement vide. Elle est implémentée physiquement sous forme de fichiers fragmentés NTFS, lesquels sont des fichiers pour lesquels l'espace disque physique est alloué uniquement lorsque cela est nécessaire. Lors de la première mise à jour d'une page de la base de données source, l'image d'origine de cette page est copiée dans la capture instantanée de la base de données. Si une page n'est jamais modifiée, elle n'est jamais copiée.

SQL Server effectue effectivement des copies de pages de données entières, même si une seule ligne a été mise à jour. L'utilisation de pages est plus efficace que l'utilisation de lignes. Les performances de lecture et d'écriture pour une page entière ou une ligne entière sont très similaires. Étant donné qu'une page peut contenir un grand nombre de lignes, si plusieurs lignes d'une page de données sont mises à jour, SQL Server ne doit effectuer qu'une seule opération de copie. Ce système de copie efficace facilite la création d'une capture instantanée de base de données même sur une base de données fréquemment mise à jour.

## 2.3.Récupération des données

Un utilisateur qui accède à une capture instantanée de base de données voit la copie d'une page dans la capture instantanée uniquement si cette page a changé depuis la création de la capture instantanée. Dans le cas contraire, l'utilisateur est redirigé vers la page correspondante dans la base de données source. Cette redirection est transparente pour l'utilisateur.

# 3. Application pratique : Création de captures instantanées de base de données

## 3.1. Création d'une capture instantanée à l'aide de Transact-SQL

Pour créer une capture instantanée de base de données à l'aide de Transact-SQL, procédez comme suit:

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, sur **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
4. Dans la nouvelle fenêtre de requête vierge, tapez le code Transact-SQL suivant.

```
CREATE DATABASE AdventureWorks_Snapshot1200 ON (
NAME = 'AdventureWorks_Data',
FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\AW_1200.ss')
AS SNAPSHOT OF AdventureWorks
```

5. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
6. Dans l'Explorateur d'objets, développez **Bases de données, Captures instantanées de base de données**, puis confirmez que la capture instantanée de base de données **AdventureWorks_Snapshot1200** existe. Vous devrez peut-être actualiser la liste des captures instantanées de la base de données pour que la capture instantanée **AdventureWorks_Snapshot1200** soit visible.
7. Cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
8. Dans la nouvelle fenêtre de requête vierge, tapez le code Transact-SQL suivant.

```
SELECT AddressID, AddressLine1, ModifiedDate FROM AdventureWorks.Person.Address
WHERE AddressID = 1
SELECT AddressID, AddressLine1, ModifiedDate FROM AdventureWorks_Snapshot1200.Person.Address
WHERE AddressID = 1
```

9. Cliquez sur le bouton **Exécuter** dans la barre d'outils, puis consultez les résultats. Les deux instructions retournent exactement les mêmes informations.
10. Cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
11. Dans la nouvelle fenêtre de requête vierge, tapez le code Transact-SQL suivant.

```
UPDATE AdventureWorks.Person.Address

SET AddressLine1 = '1000 Napa Ct.' WHERE AddressID = 1
```

12. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
13. Revenez à la fenêtre de requête précédente qui contient les instructions SELECT, puis cliquez sur le bouton **Exécuter**. Cette fois-ci, les résultats sont différents car la capture instantanée contient toujours la valeur d'origine.
14. Fermez SQL Server Management Studio. Cliquez sur **Non** lorsque vous êtes invité à enregistrer les fichiers.

# Application pratique : Création d'une base de données

# 1. Scénario

Le responsable du développement de bases de données chez Adventure Works a créé une spécification pour une nouvelle base de données dans laquelle les détails des ressources informatiques utilisées par la société seront stockés. Vous devez utiliser la spécification pour créer les groupes de fichiers appropriés, la base de données avec les options nécessaires, les schémas requis et une capture instantanée de la base de données.

Le responsable du développement de bases de données a stipulé les conditions requises suivantes pour la nouvelle base de données:

- Le nouveau nom de la base de données est **AW_ITAssets**.
- La base de données doit être composée de deux fichiers de données et d'un fichier journal portant les noms **AW_IT_Assets_Data1**, **AW_IT_Assets_Data2** et **AW_IT_Assets_Log**, respectivement. Ces fichiers doivent être créés dans le dossier C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data.
- La base de données doit utiliser deux groupes de fichiers, le groupe de fichiers primaire et un groupe de fichiers nommé **Secondary**, qui doit être le groupe de fichiers par défaut. Le fichier **AW_IT_Data2** doit être placé dans le groupe de fichiers **Secondary**.
- La taille initiale des deux fichiers de données doit être de 20 mégaoctets (Mo), et la croissance automatique n'est pas autorisée pour ces deux fichiers. La taille initiale du fichier journal doit être de 5 Mo, et la croissance automatique de ce fichier n'est pas autorisée.
- La base de données doit se fermer automatiquement lorsqu'aucun utilisateur n'est connecté, et elle doit faire l'objet d'une réduction automatique lorsque cela est utile.
- La base de données doit comporter un schéma défini par l'utilisateur nommé **TechSupport**.
- Un script permettant de créer et de remplir une table a été généré et doit être exécuté dans la base de données.
- L'utilisateur **MIAMI\Marie** doit utiliser le schéma **TechSupport** comme schéma par défaut.
- Une capture instantanée de base de données nommée **AW_IT_Snapshot** doit être créée à des fins de génération de rapports. Les fichiers de la capture instantanée de base de données doivent être créés dans le dossier C:\Program Files\Microsoft SQL Server\MSSQL. 1\MSSQL\Data\.
- Utilisez SQL Server Management Studio pour créer des scripts pour chaque action, puis enregistrez ces scripts dans un projet de scripts SQL Server dans le dossier D:\Labfiles\Starter. Ce projet peut être utilisé pour documenter la configuration de la base de données et recréer la base de données si besoin est.
- Les scripts permettant de créer des tables et des utilisateurs ont été fournis dans le dossier D:\Labfiles\Starter.

# 2. Exercice 1 : Création d'une base de données

### Création d'une nouvelle base de données avec les options et les groupes de fichiers :

| Tâche | Informations |
|---|---|
| Créer un projet de scripts SQL Server Management Studio. | 1. Créez un nouveau projet de scripts SQL Server nommé **AW_IT_Database** dans le dossier approprié. |

| | |
|---|---|
| Créer la nouvelle base de données avec les options et les groupes de fichiers appropriés. | 1. Utilisez l'Explorateur d'objets pour créer une nouvelle base de données.<br>2. Entrez les détails appropriés pour la base de données **AW_IT_Assets** et générez l'action de script vers le Presse-papiers avant de cliquer sur **OK**.<br>3. Collez le contenu du Presse-papiers dans le fichier de script **CreateAW_IT_Assets.sql**, puis enregistrez le fichier. |

## 3. Exercice 2 : Création de schéma

**Création de schémas pour la base de données AW_IT_Assets :**

| Tâche | Informations |
|---|---|
| Créer le schéma **TechSupport**. | 1. Dans le menu **Projet**, cliquez sur **Nouvelle requête** pour créer un nouveau fichier de requête. Lorsque vous y êtes invités, connectez-vous au serveur **MIAMI**.<br>2. Dans l'Explorateur de solutions, renommez SQLQuery1.sql en **CreateAW_IT_Schemas.sql**.<br>3. Utilisez l'Explorateur d'objets pour créer le schéma **TechSupport**, en générant l'action de script vers le Presse-papiers.<br>4. Collez le contenu du Presse-papiers dans le fichier de script CreateAW_IT_Schemas.sql, puis enregistrez le fichier. |
| Créer une nouvelle table et une nouvelle connexion à l'aide du script fourni. | 1. Dans le menu **Projet**, cliquez sur **Ajouter un élément existant**.<br>2. Ajoutez le fichier **D:\Labfiles\Starter\Create_AW_IT_Table_and_User.sql**. Lorsque vous y êtes invité, connectez-vous au serveur **MIAMI**.<br>3. Exécutez le script pour créer la table, la connexion et l'utilisateur. |
| Vérifier que l'utilisateur peut sélectionner des informations à partir de la table **Asset** uniquement en qualifiant le nom de la table avec le schéma. | 1. Cliquez sur **Démarrer**, sur **Exécuter**, puis exécutez la commande **runas / noprofile / user :MIAMI \Marie cmd**. Lorsque vous êtes invité à entrer le mot de passe de l'utilisateur, entrez **Pa$$w0rd**.<br>2. Dans la nouvelle fenêtre d'invite de commandes, tapez **sqlcmd - S MIAMI - E**.<br>3. Sélectionnez toutes les données de la table **Asset** à l'aide du code Transact-SQL suivant :<br>`USE AW_IT_Assets`<br>`SELECT * FROM Asset`<br>`GO`<br>Un message d'erreur s'affiche.<br>4. Sélectionnez toutes les données de la table **Asset** utilisant le schéma **TechSupport** à l'aide du code Transact-SQL suivant :<br>`USE AW_IT_Assets S`<br>`ELECT * FROM TechSupport.Asset`<br>`GO`<br>La table s'affiche correctement.<br>5. Ne fermez pas la fenêtre d'invite de commandes. |

| Tâche | Informations |
|---|---|
| Modifier les schémas par défaut des utilisateurs et tester de nouveau l'accès à la table dans le schéma par défaut. | 1. À l'aide de l'Explorateur d'objets dans SQL Server Management Studio, développez **Bases de données**, **AW_IT_Assets**, **Sécurité**, **Utilisateurs**, cliquez avec le bouton droit sur **MIAMI\Marie**, puis cliquez sur **Propriétés**.<br>2. Modifiez le schéma par défaut pour **MIAMI\Marie** en **TechSupport**, en générant l'action de script vers le Presse-papiers.<br>3. Collez le contenu du Presse-papiers dans le fichier CreateAW_IT_Schemas.sql, en ajoutant le script existant. Enregistrez ensuite tous les fichiers dans le projet.<br>4. Revenez à la fenêtre d'invite de commandes qui exécute sqlcmd, puis sélectionnez toutes les données de la table **Asset** en utilisant le schéma par défaut, comme indiqué dans le code Transact-SQL suivant :<br>`USE AW_IT_Assets`<br>`SELECT * FROM Asset`<br>`GO`<br>La table s'affiche correctement.<br>5. Quittez sqlcmd et fermez la fenêtre d'invite de commandes. |

## 4. Exercice 3 : Création d'une capture instantanée de base de données

**Création d'une capture instantanée de la base de données AW_IT_Assets :**

| Tâche | Informations |
|---|---|
| Créer une capture instantanée de base de données, puis comparer la taille des fichiers de données de la capture instantanée avant et après la modification des données. | 1. Ajoutez un nouveau fichier de requête au projet, en vous connectant au serveur MIAMI lorsque vous y êtes invité.<br>2. Renommez le nouveau fichier de requête en **CreateAW_IT_Snapshot.sql**.<br>3. Ajoutez le code Transact-SQL pour créer une nouvelle capture instantanée de base de données nommée **AW_IT_Assets_Snapshot1** reposant sur la base de données **AW_IT_Assets** en utilisant l'instruction Transact-SQL CREATE DATABASE. Vous devez créer deux fichiers de capture instantanée pour établir la correspondance avec les deux fichiers de données de la base de données source. Placez les deux fichiers de données des captures instantanées dans C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data.<br>4. Affichez la taille des fichiers de capture instantanée à l'aide de l'Explorateur Windows et observez la valeur **Taille sur le disque** actuelle.<br>5. Modifiez les données dans la base de données source en insérant un nouvel enregistrement dans la table **Asset**, puis vérifiez que le nouvel enregistrement existe dans la base de données source mais pas dans la capture instantanée de base de données.<br>6. Affichez la nouvelle valeur **Taille sur le disque** pour le deuxième fichier de données de la capture instantanée à l'aide de l'Explorateur Windows. Cette valeur a augmenté à cause de la modification des données. |

## Liste de contrôle des résultats :

Utilisez la liste de contrôle des résultats ci-après pour vérifier que vous avez correctement effectué cet atelier pratique:

- Création d'une base de données nommée **AW_IT_Assets**.
- Configuration d'un groupe de fichiers nommé **Secondary** qui contient un fichier de données secondaire.
- Création d'un schéma **TechSupport**.
- Affectation de **TechSupport** comme schéma par défaut du compte d'utilisateur **MIAMI\Marie**.
- Création d'une capture instantanée de la base de données nommée **AW_IT_Assets_Snapshot1**.

# MODULE 2 : CREATION DE TYPES DE DONNEES ET DE TABLES

## Leçon 1 : Création de types de données

### 1. Que sont les types de données par le système ?

| Catégorie | | Types de données |
|---|---|---|
| Numérique | Entier | int, bigint, smallint, tinyint |
| | Exact | decimal, numeric |
| | Approximatif | float, real |
| | Monétaire | money, smallmoney |
| Date et heure | | datetime, smalldatetime |
| Caractère | Non-Unicode | char, varchar, varchar(max), text |
| | Unicode | nchar, nvarchar, nvarchar(max), ntext |
| Binaire | | binary, varbinary, varbinary(max) |
| Image | | image |
| Identificateur global | | uniqueidentifier |
| XML | | xml |
| Spécial | | bit, cursor, timestamp, sysname, table, sql_variant |

#### 1.1Introduction

Les types de données d'une colonne définissent les valeurs de données que celle-ci accepte dans une table de base de données. SQL Server fournit plusieurs types de données. Certaines catégories de types de données fréquemment adoptées dans les langages de programmation disposent de plusieurs types de données SQL Server qui leurs sont associés. Vous devez utiliser le plus petit type de données répondant à vos besoins. Vous économiserez ainsi de l'espace disque et permettrez l'insertion du plus grand nombre de lignes possible sur une page de données, ce qui vous garantira des performances optimales.

#### 1.2 Types de données fournis par le système

Le tableau ci-dessous décrit la base de données fournie par le système dans SQL Server 2005 et indique les synonymes ANSI (American National Standards

Institute) des types de données standard utilisés sur tous les systèmes de base de données conformes ANSI. Lorsque vous référencez des types de données en code Transact-SQL, vous pouvez utiliser le nom de type de données SQL Server spécifique ou le synonyme ANSI.

| Catégorie de type de données | Types de données fournis par le système SQL Server | Synonyme ANSI | Nombre d'octets |
|---|---|---|---|
| Entier | **int**<br>**bigint**<br>**smallint, tinyint** | *integer*<br>–<br>– | 4 8 2, 1 |
| Valeur numérique exacte | **decimal[(p[, s])]**<br>**numeric[(p[, s])]** | *dec*<br>– | 2–17 |
| Valeur numérique approximative | **float[(n)] real** | *double precision, float[(n)] for n=8–15 float[(n)] for n=1–7* | 8 4 |
| Monétaire | **money, smal l money** | – | 8, 4 |
| Date et heure | **datetime, smalldatetime** | – | 8, 4 |
| Caractère non-Unicode | **char[(n)]**<br>**varchar[(n)]**<br>**varchar(max)**<br>**text** | *character[(n)]*<br>*char VARYING[(n)]*<br>*character VARYING[(n)] char VARYING(max)*<br>*character VARYING(max)* – | 0–8,000 0–8,000<br>0 à 2 gigaoctets (Go) (dans la ligne ou le pointeur de 16 octets)<br>0 à 2 Go (pointeur de 16 octets) |
| Caractère Unicode | **nchar [(n)]**<br>**nvarchar [(n)]**<br>**nvarchar ( max )**<br>**ntext** | *national char[(n)]*<br>*national chara c ter[(n)] national char VARYING([n])*<br>*national character VARYING([n])*<br>*national char VARYING(max)*<br>*national character VARYING(max)* – | 0 à 8 000 (4 000 caractères)<br>0 à 8 000 (4 000 caractères)<br>0 à 2 Go (dans la ligne ou le pointeur de 16 octets)<br>0 à 2 Go (pointeur de 16 octets) |
| Binaire | **binary[(n)]**<br>**varbinary [(n)]**<br>**varbinary (max)** | –<br>*binary VARYING[(n)]*<br>*binary VARYING(max)* | 0–8,000 0–8,000<br>0 à 2 Go (dans la ligne ou le pointeur de 16 octets) |
| Image | **image** | – | 0 à 2 Go (pointeur de 16 octets) |
| Identificateur global | **uniqueidentifier** | – | 16 |
| XML | **xml** | – | 0 à 2 Go |
| Spécial | **bit, cursor**<br>**timestamp**<br>**sysname table**<br>**sql_variant** | – *rowversion*<br>–<br>–<br>– | 1, 0–8 8 256<br>0–8,016 |

## 1.3Types de données de longueur fixe et variable

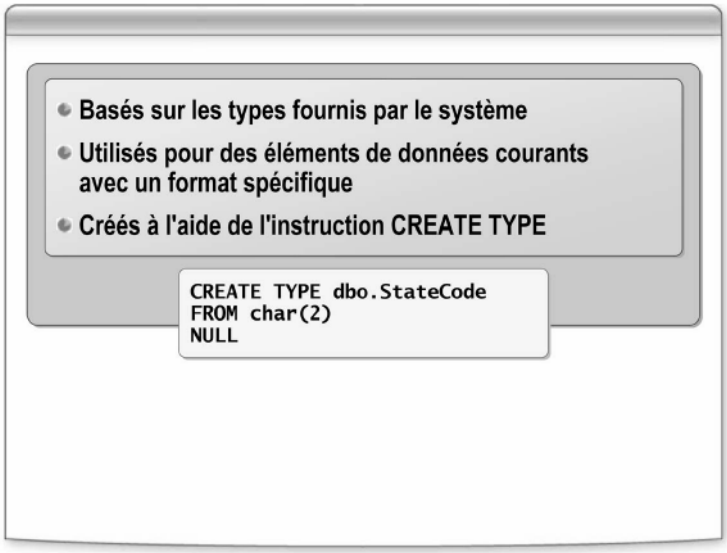
Lorsque vous savez que toutes les valeurs d'une entité de données spécifique auront exactement la même taille, vous pouvez utiliser un type de données de longueur fixe

(par exemple, nchar) et indiquer le nombre d'octets à réserver pour chaque valeur. Lorsque la longueur de la valeur est susceptible de varier, il peut être plus efficace d'adopter un type de données de longueur variable comme nvarchar et de spécifier la longueur maximale de la valeur. Lorsque vous spécifiez un type de données de longueur variable, SQL Server réserve 2 octets pour chaque valeur de longueur variable en tant que marqueur et utilise uniquement l'espace supplémentaire requis pour chaque valeur.

### 1.4 Valeurs de données élevées

Les colonnes employées pour le stockage de valeurs de données de taille moyenne à élevée (généralement supérieures à 8 000 octets) peuvent être stockées à l'aide d'un type d'objet volumineux (LOB), tel que text ou image, ou en tant que colonne varchar, nva r char ou varbinary déclarée avec le spécificateur max. Les types de données LOB sont fournis à l'origine pour des raisons de compatibilité descendante. En règle générale, vous devez utiliser le spécificateur max pour le stockage de valeurs de données élevées.

## 2. Que sont les types de données d'alias ?




### 2.1. Types de données d'alias

Un type de données d'alias est un type de données personnalisé basé sur un type de données fourni par le système. Il présente les caractéristiques suivantes:

- Il vous permet d'affiner davantage les types de données pour garantir une cohérence lors de l'utilisation d'éléments de données communs dans différentes tables ou bases de données.
- Il est défini dans une base de données spécifique.
- Il doit disposer d'un nom unique dans la base de données. (Mais les types de données d'alias portant des noms différents peuvent avoir la même définition.)

### 2.2. Quand créer des types de données d'alias

Vous devez créer un type de données d'alias lorsque vous devez définir un élément de données fréquemment employé avec un format spécifique. Par exemple, une colonne dans laquelle vous stockez un code de pays fondé sur la norme alpha-2 des abréviations des noms de pays (par exemple, JP pour le Japon et CH pour la Suisse) de l'Organisation internationale de normalisation (ISO) peut être définie sous la forme **char(2)**. Toutefois, si le code de pays est destiné à un usage fréquent dans la base de données, vous pouvez définir un type de données **Countr y Code** (CodePays) et l'utiliser à la place. Cela facilite la compréhension des définitions d'objets et du code dans votre base de données.

### 2.3. Exemple de création d'un type de données d'alias

Vous pouvez créer des types de données d'alias à l'aide de l'Explorateur d'objets dans SQL Server Management Studio ou de l'instruction Transact-SQL CREATE TYPE. L'exemple de code suivant explique comment créer un type de données d'alias nommé **CountryCode** (CodePays).

```
CREATE TYPE dbo.CountryCode
FROM char(2)
 NULL
```

Notez que lorsque vous créez un type de données d'alias, vous pouvez spécifier sa possibilité de valeur NULL. Un type de données d'alias créé avec l'option NOT NULL ne peut jamais être utilisé pour le stockage d'une valeur NULL. Il est important de spécifier la possibilité de valeur NULL appropriée lorsque vous créez un type de données. Pour modifier un type de données, vous devez utiliser l'instruction DROP TYPE pour supprimer le type de données, puis recréer un nouveau type de données afin de le remplacer. Du fait que vous ne pouvez pas supprimer un type de données qui est utilisé par les tables de la base de données, vous devez également MODIFIER chaque table qui exploite en premier le type de données.

# 3. Application pratique : Création de type de données

## 3.1. Création d'un type d'alias à l'aide de SQL Server Management Studio

Procédez comme suit pour créer un type d'alias à l'aide de SQL Server Management Studio:

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes** et **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Si l'Explorateur d'objets n'apparaît pas, cliquez sur **Explorateur d'objets** dans le menu **Affichage**.
4. Dans l'Explorateur d'objets, développez **Bases de données**, **AdventureWorks**, **Programm a bilité** et **Types**.
5. Cliquez avec le bouton droit sur **Types de données définis par l'utilisateur**, puis cliquez sur **Nouveau type de données défini par l'utilisateur**.
6. Dans la boîte de dialogue **Nouveau type de données défini par l'utilisateur**, entrez les détails dans le tableau suivant, puis cliquez sur **OK**.

| Propriété | Valeur |
|---|---|
| Schéma | dbo |
| Nom | CountryCode |
| Type de données | char |
| Longueur | 2 |
| Autoriser les valeurs NULL | Option activée |

7. Vérifiez que le type de données CountryCode apparaît dans la liste Types de données définis par l'utilisateur.
8. Laissez SQL Server Management Studio ouvert. Vous allez l'utiliser au cours de la procédure suivante.

## 3.2. Création d'un type d'alias à l'aide de Transact-SQL

Procédez comme suit pour créer un type de données défini par l'utilisateur à l'aide de Transact-SQL:

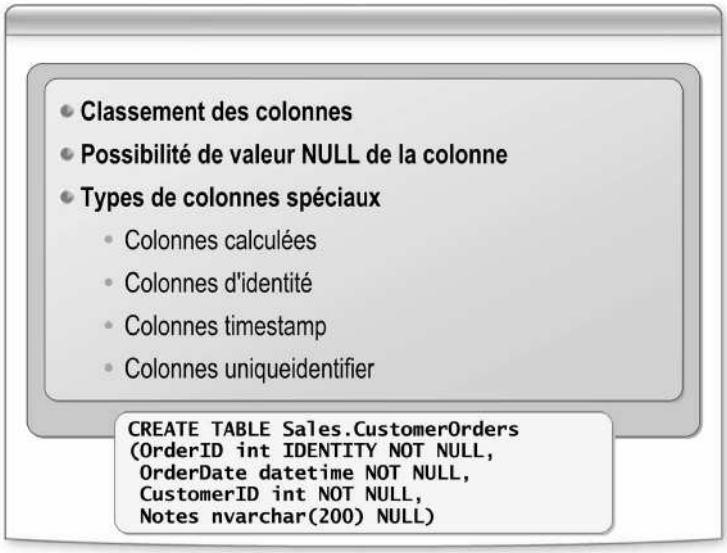
1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
2. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks
CREATE TYPE dbo.EmailAddress FROM nvarchar(50)
NULL
```

3. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
4. Une fois l'exécution réussie, cliquez avec le bouton droit sur le dossier **Types de données définis par l'utilisateur** dans l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que le type de données **dbo.EmailAddress** a été ajouté dans la base de données.
5. Fermez SQL Server Management Studio. Cliquez sur **Non** si le programme vous demande d'enregistrer tous les fichiers.

# Leçon 2 : Création de tables

## 1. Eléments à prendre en compte pour la création de tables




### 1.1 Introduction

Lorsque vous créez une table, vous devez spécifier le nom de la table ainsi que les noms et les types de données des colonnes. Les noms de colonnes doivent être uniques à une table donnée. Toutefois, vous pouvez utiliser le même nom de colonne dans différentes tables au sein de la même base de données. Vous devez spécifier un type de données pour chaque colonne. Vous pouvez disposer des éléments suivants:

- Plus de 2 milliards d'objets par base de données, tables comprises.
- Jusqu'à 1 024 colonnes par table.
- Jusqu'à 8 060 octets par ligne. (La longueur maximale approximative ne s'applique pas aux types de données **image**, **text** et **ntext** ou aux colonnes définies à l'aide du spécificateur **max**.)

### 1.2 Caractéristiques

| | |
|---|---|
| **Possibilité de valeur NULL** | Vous pouvez préciser dans la définition de table si l'utilisation de valeurs NULL dans chaque colonne est autorisée. Si vous ne spécifiez pas l'option NULL ou NOT NULL, SQL Server attribue la caractéristique NULL ou NOT NULL en fonction de la valeur par défaut au niveau de la session ou de la base de données. Toutefois, ne vous fiez pas à ces valeurs par défaut car elles peuvent changer. |

**Types de colonnes spéciaux**

Les types spéciaux de colonnes incluent les éléments suivants :

*Colonnes calculées.* Une *colonne calculée* est une colonne virtuelle qui n'est pas stockée physiquement dans la table. SQL Server utilise une formule que vous créez pour calculer cette valeur de colonne à l'aide d'autres colonnes dans la même table. L'utilisation d'un nom de colonne calculée dans une requête peut simplifier la syntaxe de la requête.

*Colonnes d'identité.* Vous pouvez utiliser la propriété **Identity** pour créer des colonnes (appelées *colonnes d' identité*) qui contiennent des valeurs séquentielles générées par le système et identifiant chaque ligne insérée dans une table. Une colonne d'identité est souvent utilisée pour les valeurs de clé primaire. En demandant à SQL Server de fournir automatiquement les valeurs de clés, vous pouvez réduire vos coûts et améliorer vos performances. Le travail de programmation est simplifié, les valeurs de clé primaire restent limitées en taille et le risque de goulots d'étranglement découlant des transactions des utilisateurs est réduit.

*Colonnes timestamp.* Les colonnes définies avec le type de données **timestamp** affiche une valeur par défaut qui désigne un horodatage généré automatiquement et garanti comme unique au sein de la base de données.

*Colonnes uniqueidentifier.* Les colonnes définies avec le type de données **uniqueidentifier** peuvent être utilisées pour le stockage d'identificateurs uniques globaux (GUID) garantis universellement uniques. Vous pouvez générer une valeur pour une colonne **uniqueidentifier** à l'aide de la fonction Transact-SQL NEWID.

**Exemple de création d'une table**

Vous pouvez créer des tables à partir de l'Explorateur d'objets dans SQL Server Management Studio ou à l'aide de l'instruction Transact-SQL CREATE TABLE. L'exemple de code Transact-SQL suivant peut être utilisé pour créer une table nommée **CustomerOrders** (CommandesClients) dans un schéma appelé **Sales** (Ventes). La table comporte une colonne d'identité.

```
CREATE TABLE Sales.CustomerOrders
(OrderID int identity NOT NULL,
 OrderDate datetime NOT NULL,
 CustomerID int NOT NULL,
 Notes nvarchar(200) NULL)
```

Vous pouvez également créer une table à l'aide des outils de l'Explorateur d'objets dans SQL Server Management Studio.

**Modification et suppression de tables**

Vous pouvez modifier une définition de table à partir de l'Explorateur d'objets ou à l'aide de l'instruction Transact-SQL ALTER TABLE. Par exemple, les instructions Transact-SQL suivantes indiquent comment ajouter une colonne et modifier la possibilité de valeur NULL d'une colonne dans la table **Sales.CustomerOrders** définie précédemment.

```
ALTER TABLE Sales.CustomerOrders
ADD SalesPersonID int NOT NULL
GO
ALTER TABLE Sales.CustomerOrders
ALTER COLUMN Notes nvarchar(200) NOT NULL
GO
```

Vous pouvez supprimer une table de la base de données à partir de l'Explorateur d'objets ou à l'aide de l'instruction Transact-SQL DROP TABLE.

# 2. Application pratique : Création de tables

## 2.1. Création d'une table à l'aide de SQL Server Management studio

Procédez comme suit pour créer une table à l'aide de SQL Server Management Studio:

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes** et **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.

2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.

3. Si l'Explorateur d'objets n'apparaît pas, cliquez sur **Explorateur d'objets** dans le menu **Affichage**.
4. Dans l'Explorateur d'objets, développez **Bases de données**, **Adventur e Works** et **Tables**.

5. Cliquez avec le bouton droit sur **Tables**, puis cliquez sur **Nouvelle table**.

6. Dans la fenêtre **Table – dbo.Table_1**, entrez les détails suivants:

| Nom de la colonne | Type de données | Autoriser les valeurs NULL |
|---|---|---|
| DiscountID | int | Option non activée |
| Amount | decimal (18, 0) | Option non activée |
| DiscountName | nvarchar (50) | Option non activée |
| Description | nvarchar (MAX) | Option activée |

7. Cliquez sur la ligne **DiscountID**, puis examinez le volet **Propriétés des colonnes** sous la liste des colonnes.

8. Développez **Spécification du compteur**, puis définissez la propriété **(Est d'identité)** à **Oui**.

9. Dans le menu **Concepteur de tables**, cliquez sur **Définir la clé primaire**.

10. Cliquez sur la ligne **DiscountName**, puis examinez le volet **Propriétés des colonnes**.

11. Définissez la propriété **Longueur** à **25**.

Dans le menu **Affichage**, cliquez sur **Fenêtre Propriétés**, puis définissez la propriété **Schéma** à **Sales** (Ventes).

12. Cliquez n'importe où dans la surface de création de la table, puis dans le menu **Fichier**, cliquez sur **Enregistrer Table_1**.

13. Dans la boîte de dialogue **Choisir un nom**, tapez **Discount**, puis cliquez sur **OK**.

14. Fermez la fenêtre **Table- Sales.Discount**, cliquez avec le bouton droit sur le dossier **Tables**, puis cliquez sur **Actualiser** pour vérifier que la nouvelle table a été créée.

15. Laissez SQL Server Management Studio ouvert. Vous allez l'utiliser au cours de la procédure suivante.

## 2.2.Création d'une table à l'aide de Transct-SQL

Procédez comme suit pour créer une table à l'aide de Transact-SQL:

1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.

2. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks
CREATE TABLE Sales.SpecialOffers
(SpecialOfferID int IDENTITY PRIMARY KEY NOT NULL,
 OfferName nvarchar(25) NOT NULL,
 Description nvarchar(max) NULL)
```

3. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
4. Une fois l'exécution réussie, cliquez avec le bouton droit sur le dossier **Tables** dans l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que la table **Sales.SpecialOffers** a été ajoutée dans la base de données.

# Atelier pratique: Création de types de données et de tables

## 1. Scénario

Le directeur commercial d'Adventure Works souhaite que la base de données de la société soit modifiée afin d'y inclure des données sur les marchandises retournées et les ventes remboursées. Le développeur principal de la base de données a conçu quelques nouveaux types de données et tables pour les retours et les remboursements et vous a confié la tâche de les créer dans la base de données. Les critères requis pour la modification de la base de données incluent les points suivants:

- Vous devez utiliser SQL Server Management Studio pour créer un projet de script SQL Server pour la solution dans le dossier D:\Labfiles\Starter.
- Vous devez créer un nouveau type de données appelé **ShortDescription** dans le schéma **dbo**. Ce type de données doit être basé sur le type de données fourni par le système **nvarchar** et avoir une longueur maximale de 100 caractères.
- Vous devez créer un nouveau type de données appelé **CashValue** dans le schéma **dbo**. Ce type de données doit être basé sur le type de données fourni par le système **decimal** et doit avoir une précision de 8 et une échelle de 2.
- Vous devez créer une table appelée **ReturnedGoods** dans le schéma **Sales**. La table **ReturnedGoods** doit contenir les colonnes suivantes:
- **ReturnID.** Colonne d'identité **int** ne pouvant contenir aucune valeur NULL.
- **ProductID.** Colonne **int** ne pouvant contenir aucune valeur NULL.
- **CustomerID.** Colonne **int** ne pouvant contenir aucune valeur NULL.
- **ReturnDate.** Colonne **datetime** ne pouvant contenir aucune valeur NULL.
- **ReturnReason.** Colonne **ShortDescription** pouvant contenir des valeurs NULL.

- Vous devez créer une table appelée **Refunds** dans le schéma **Sales**. La table **Refunds** doit contenir les colonnes suivantes:
- **RefundID.** Colonne d'identité **int** ne pouvant contenir aucune valeur NULL.
- **ReturnID.** Colonne **int** ne pouvant contenir aucune valeur NULL.
- **Amount.** Colonne **CashValue** ne pouvant contenir aucune valeur NULL.
- Dans le schéma **Sales**, vous devez créer une table partitionnée appelée **ReturnsArchive** contenant les retours archivés dans la même structure que la table **ReturnedGoods** mais stockant les données dans trois partitions : une pour les retours enregistrés avant 2005, une pour les retours enregistrés courant 2005 et une pour les retours enregistrés en 2006.
- Un script permettant de renseigner la table **ReturnsArchive** est fourni dans le dossier D:\Labfiles\Starter.

## 2. Autres informations

Lorsque vous effectuez des tâches de développement de base de données, il peut être utile de faire appel à SQL Server Management Studio pour créer un projet de script SQL Server et l'utiliser pour documenter le code Transact-SQL nécessaire pour recréer la solution si besoin est.

Utilisez la procédure suivante pour créer un projet de script SQL Server :

1. Ouvrez SQL Server Management Studio en vous connectant au serveur que vous souhaitez gérer.

2. Dans le menu **Fichier**, pointez sur **Nouveau**, puis cliquez sur **Projet**.

3. Sélectionnez le modèle **Scripts SQL Server** et entrez un nom et un emplacement appropriés pour le projet. Notez que vous pouvez créer une solution contenant plusieurs projets mais que, dans nombre de cas, un seul projet par solution convient.

Pour ajouter un fichier de requête à un projet:

1. Cliquez sur **Nouvelle requête** dans le menu **Projet** ou cliquez avec le bouton droit sur le dossier **Requêtes** dans l'Explorateur de solutions, puis cliquez sur **Nouvelle requête**. Si l'Explorateur de solutions n'apparaît pas, cliquez sur **Explorateur de solutions** dans le menu **Affichage** pour l'afficher.

2. Lorsque le système vous le demande, connectez-vous au serveur sur lequel vous voulez exécuter la requête. Un objet de connexion est alors ajouté au projet.

3. Pour modifier le nom par défaut du fichier de requête (**SQLQuery1.sql**), cliquez dessus avec le bouton droit dans l'Explorateur de solutions, puis cliquez sur **Renommer**.

Bien qu'il vous soit possible d'effectuer l'ensemble des tâches de développement de base de données en exécutant des instructions Transact-SQL, il est souvent plus facile de recourir à l'interface utilisateur graphique de SQL Server Management Studio. Toutefois, vous devez générer les scripts Transact-SQL correspondants et les enregistrer dans le projet pour référence ultérieure.

Souvent, vous pouvez générer le script Transact-SQL pour une action avant de cliquer sur **OK** dans la boîte de dialogue **Propriétés** utilisée pour réaliser l'action. Beaucoup de boîtes de dialogue **Propriétés** incluent une liste déroulante **Script** par le biais de laquelle vous pouvez générer une action de script pour une nouvelle fenêtre de requête, un fichier, le Presse-papiers ou un travail de SQL Server Agent. Une technique courante consiste à ajouter un fichier de requête vide à un projet, à générer une action de script pour le Presse- papiers tel qu'il est exécuté, puis à coller le script généré dans le fichier de requête.

Vous pouvez également générer des scripts pour un grand nombre d'objets existants, notamment des bases de données et des tables. Pour générer un script, cliquez avec le bouton droit sur l'objet dans l'Explorateur d'objets et générez l'action de script CREATE. Si l'Explorateur d'objets n'apparaît pas, cliquez sur **Explorateur d'objets** dans le menu **Affichage** pour l'afficher.

# 3. Exercice 1 : Création de types de données

## Création des types de données ShortDescription et CashValue

| Tâche | Informations |
|---|---|
| Créer un projet de script SQL Server. | 1. Démarrez SQL Server Management Studio.<br>2. Créez un nouveau projet de script SQL Server nommé **AW_Tables**. |
| Créer les types de données **ShortDescription** et **CashValue**. | 1. Ajoutez un nouveau fichier de requête au projet en vous connectant à **MIAMI** lorsque le système vous y invite. Renommez le fichier de requête **CreateDataTypes.sql**.<br>2. Dans la fenêtre de la requête, tapez l'instruction Transact-SQL appropriée pour créer le type de données **ShortDescri p tion** dans la base de données **AdventureWorks**.<br>3. Ajoutez une deuxième instruction Transact-SQL pour créer le type de données **CashValue** dans la base de données **AdventureWorks**.<br>4. Exécutez la requête, puis enregistrez le fichier de requête.<br>5. Utilisez l'Explorateur d'objets pour vérifier que les types de données ont été créés.<br>6. Laissez SQL Server Management Studio ouvert. Vous allez l'utiliser au cours de l'exercice suivant. |

# 4. Exercice 2

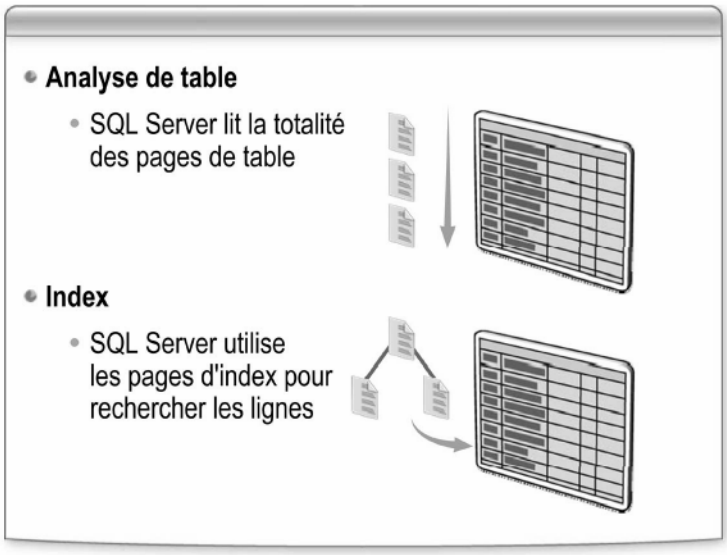
### Création des tables ReturnedGoods et Refunds

| Tâche | Informations |
|---|---|
| Créer un nouveau fichier de requête. | 1. Ajoutez un nouveau fichier de requête au projet en vous connectant à **MIAMI** lorsque le système vous y invite.<br>2. Renommez le fichier de requête **CreateTables.sql**. |
| Créer les tables **ReturnedGoods** et **Refunds**. | 1. Dans la fenêtre de la requête, tapez l'instruction Transact-SQL appropriée pour créer la table **ReturnedGoods** dans la base de données **AdventureWorks**.<br>2. Ajoutez une deuxième instruction Transact-SQL pour créer la table **Refunds**.<br>3. Exécutez la requête, puis enregistrez le fichier de requête.<br>4. Utilisez l'Explorateur d'objets pour vérifier que les tables ont été créées.<br>5. Laissez SQL Server Management Studio ouvert. Vous allez l'utiliser au cours de l'exercice suivant. |

# MODULE 3 : CREATION ET PARAMETRAGE DES INDEX

## Leçon 1 : Planification des index

### 1. Comment SQL Server accède aux données ?

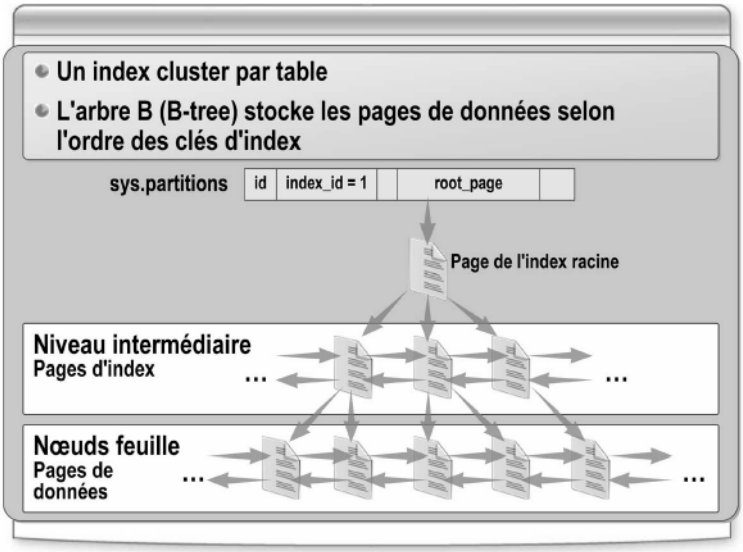



SQL Server peut accéder aux données de deux façons :

- En analysant l'ensemble des pages de données d'une table : cette analyse est appelée *analyse de table*. Lorsque SQL Server exécute une analyse de table, il :
  1. démarre en début de table ;
  2. analyse page après page toutes les lignes de la table ;
  3. extrait les lignes qui répondent aux critères de la requête.
- En utilisant les index. Lorsque SQL Server utilise un index, il :
  1. parcourt l'arborescence des index pour trouver les lignes demandées par la requête ;
  2. extrait uniquement les lignes nécessaires qui répondent aux critères de la requête.

SQL Server détermine d'abord si un index existe. Puis, l'optimiseur de requête – composant chargé de générer le plan d'exécution optimal d'une requête – détermine si la solution la plus efficace pour accéder aux données consiste à analyser une table ou à utiliser l'index.

### 2. Qu'est-ce qu'un index cluster ?




Un index cluster trie et stocke les lignes de données de la table selon un ordre fondé sur la clé d'index cluster. L'index cluster est implémenté en tant qu'arbre B (B-tree). Chaque page d'un arbre B (B-tree) est appelée nœud d'index. Le nœud supérieur de l'arbre B (B-tree) est appelé nœud racine. Le niveau inférieur des nœuds de l'index est appelé niveau feuille. Tous les niveaux d'index compris entre le nœud racine et les nœuds feuille sont appelés niveaux intermédiaires. Chaque page d'un niveau intermédiaire ou inférieur possède un pointeur vers les pages précédentes et suivantes, constituant ainsi une liste à double liaison. Cette structure fournit un mécanisme extrêmement efficace pour accélérer la recherche des données.

Dans un index cluster, le nœud racine et les nœuds intermédiaires contiennent des pages d'index composées de lignes d'index. Chaque ligne d'index contient une valeur de clé et un pointeur vers une page de niveau intermédiaire de l'arbre B (B-tree) ou vers une ligne de données du niveau feuille de l'index. Les pages de chaque niveau de l'index sont liées dans une liste à double liaison.

Comme un index cluster détermine l'ordre dans lequel les lignes d'une table sont réellement stockées, chaque table ne peut avoir qu'un seul index cluster – il n'est pas possible de stocker les lignes d'une table selon différents ordres.

**Important** Les colonnes ayant les types de données suivants ne peuvent pas être utilisées comme clé d'un index cluster : **ntext**, **text**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)** ou **image**.

## 2.1. Quand utiliser un index cluster ?

Comme vous ne pouvez avoir qu'un seul index cluster par table, assurez-vous que vous l'utilisez de façon à en retirer le plus grand bénéfice possible. Avant de créer un index cluster, vous devez comprendre comment se déroule l'accès aux données. Un index cluster déterminant l'ordre dans lequel SQL Server 2005 stocke les lignes de données de la table, il est plus adapté à certains types de données et modèles d'utilisation.

Un index cluster est très efficace quand il est utilisé pour prendre en charge les requêtes qui exécutent les actions suivantes:

- Retourner une plage de valeurs à l'aide d'opérateurs tels que BETWEEN, >, >=, < et <=. Comme les données des tables sont stockées dans l'ordre de l'index, dès que la ligne comportant la première valeur est trouvée à l'aide de l'index cluster, les lignes contenant les valeurs indexées suivantes sont nécessairement adjacentes.
- Retourner les données triées à l'aide de la clause ORDER BY ou GROUP BY. Si un index est présent sur

les colonnes spécifiées dans la clause ORDER BY ou GROUP BY, le moteur de base de données n'a plus besoin de trier les données, car les lignes le sont déjà. Les performances des requêtes s'en trouvent améliorées.

- Retourner les données associées à l'aide de clauses JOIN – il s'agit généralement de colonnes de clé étrangère.
- Retourner les jeux de résultats volumineux.

### 2.2. Elément à prendre en compte pour l'utilisation d'un index cluster ?

Lorsque vous définissez un index cluster, vous devez définir la clé d'index avec aussi peu de colonnes que possible. Le fait que la clé cluster ait une valeur réduite augmente le nombre de lignes d'index qui peuvent être placées sur une page d'index et diminue le nombre de niveaux à parcourir. Les entrées/sorties (E/S) s'en trouvent réduites.
De même, pensez à employer les colonnes qui présentent une ou plusieurs des caractéristiques ci-après :

- La colonne est unique ou contient de nombreuses valeurs distinctes, notamment une colonne définie en tant qu'IDENTITY, son unicité au sein de la table étant garantie. L'unicité de valeur de clé est gérée explicitement, à l'aide du mot clé UNIQUE, ou implicitement, à l'aide d'un identificateur unique interne. Si un index cluster contient des valeurs dupliquées, SQL Server doit repérer les lignes qui contiennent des valeurs identiques dans la ou les colonnes de clé. À cette fin, il utilise un entier de 4 octets (valeur uniquifier) dans une autre colonne uniquifier du système uniquement. Ces identificateurs uniques sont internes à SQL Server et l'utilisateur ne peut pas y accéder.
- La colonne est fréquemment utilisée pour trier les données extraites d'une table, car elle économise ainsi une opération de tri chaque fois qu'une requête trie les résultats sur la colonne.
- L'accès à la colonne se déroule généralement de façon séquentielle.

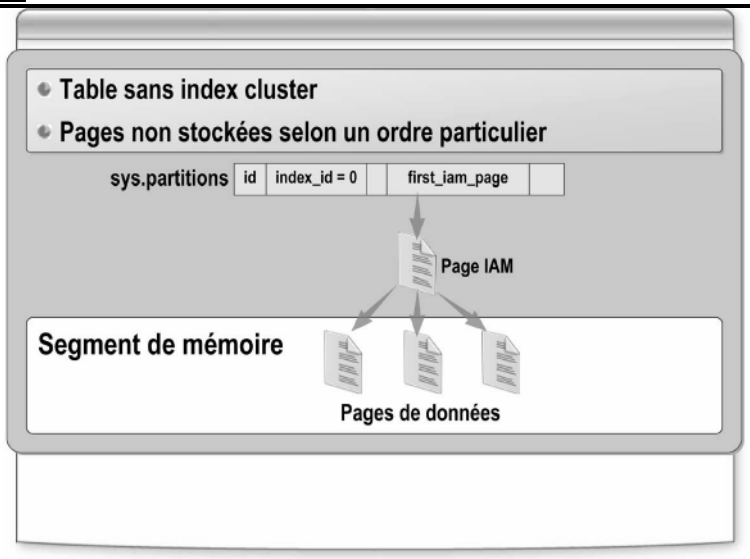
### 2.3. Quand ne pas utiliser les index cluster ?

Les index cluster ne sont pas un bon choix dans les cas suivants:

- Les données des colonnes indexées sont souvent modifiées. Les modifications apportées à un index cluster signifient que toute la ligne de données doit être déplacée, le moteur de base de données devant conserver les valeurs des données d'une ligne dans l'ordre physique. Cet élément est important dans les systèmes de traitement transactionnel à haut volume où les données sont en général éphémères.
- Les clés d'index sont étendues. Les clés étendues sont composées de plusieurs colonnes ou de plusieurs colonnes de grande taille. Tous les index non-cluster utilisent les valeurs de clés de l'index cluster comme clés de recherche. Tous les index non-cluster définis sur la même table sont considérablement plus grands, car leurs entrées contiennent la clé de cluster, ainsi que les colonnes clés définies pour cet index non-cluster.

## 3. Qu'est-ce qu'un segment de mémoire ?

### 3.1. Définition




Un segment est une table sans index cluster. Les lignes de données et la séquence des pages de données ne sont pas stockées selon un ordre particulier. Les pages de données ne sont pas associées dans une liste liée. SQL Server gère toujours les pages de données dans un segment de mémoire à moins qu'un index cluster ne soit défini sur la table.

SQL Server utilise les pages IAM pour gérer les segments de mémoire. Les pages IAM:

- Contiennent des informations sur l'emplacement où SQL Server stocke les extensions d'un segment de mémoire. La table système **sys.partitions** stocke un pointeur vers la première page IAM associée à un segment de mémoire. Il s'agit d'un enregistrement dont index_id = 0.
- Permettent la navigation dans le segment de mémoire pour rechercher l'espace disponible quand de nouvelles lignes sont insérées dans la table.
- Associent les pages de données à la table. Les pages de données et les lignes qu'elles contiennent ne sont ni classées dans un ordre spécifique ni liées. La seule association logique entre les pages de données concerne les informations enregistrées dans les pages IAM.

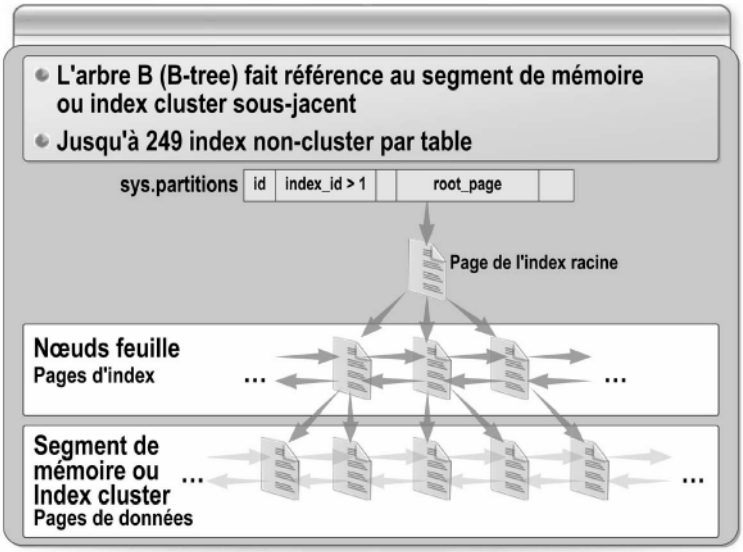
### 3.2. Quand utiliser un segment de mémoire ?

Vous utilisez par défaut un segment de mémoire chaque fois que vous ne définissez pas d'index cluster sur une table. Choisissez d'utiliser un segment de mémoire lorsque la table contient des données qui sont structurées ou utilisées d'une façon non appropriée à l'implémentation d'un index cluster. Vous pouvez toujours implémenter des index non-cluster sur une table qui utilise un segment de mémoire.

Pensez à utiliser un segment de mémoire pour les tables qui:

- Contiennent des données éphémères où l'ajout, la suppression et la mise à jour des lignes constituent des opérations fréquentes. Le temps nécessaire à la maintenance d'un index risque d'annuler les avantages que vous êtes susceptible d'en retirer.
- Contiennent de faibles volumes de données. Le recours à une analyse de table peut se révéler plus rapide que l'utilisation d'un index et sa maintenance.
- Contiennent en majorité des lignes de données dupliquées. Une analyse de table peut être plus rapide qu'une recherche d'index.
- Contiennent des données qui sont écrites mais rarement lues, comme dans le cas d'un journal d'audit. Un index peut représenter un stockage inutile et une charge de maintenance supplémentaire

## 4. Qu'est-ce qu'un index non cluster ?




Les index non-cluster ont la même structure d'arbre B (B-tree) que les index cluster, à la différence que les lignes de données de la table sous-jacente ne sont pas triées et stockées suivant l'ordre des clés non-cluster. Dans l'index non-cluster, les données et l'index sont stockés séparément, et le niveau feuille de l'index se compose de pages d'index et non de pages de données.

Les lignes de l'index non-cluster sont stockées selon l'ordre des valeurs clés de l'index. L'ordre spécifique des lignes de données référencées n'est garanti que si un index cluster est créé sur la table. Chaque ligne d'index de l'index non-cluster contient la valeur de clé non-cluster et un localisateur de ligne. Ce localisateur pointe sur la ligne de données si la table est un segment de mémoire et contient la clé d'index cluster de la ligne si la table possède un index cluster.

Si la table indexée possède un index cluster, la ou les colonnes définies dans cet index sont automatiquement ajoutées à la fin de chaque index non-cluster de la table. Il est ainsi possible de produire une requête couverte sans spécifier les colonnes de l'index cluster dans la définition de l'index non-cluster. Par exemple, si une table possède un index cluster sur la colonne **C**, un index non-cluster sur les colonnes **B** et **A** aura comme valeurs de clés les colonnes **B**, **A** et **C**.

Une table peut avoir jusqu'à 249 index non-cluster. Les index non-cluster peuvent être définis sur une table, que celle-ci utilise un index cluster ou un segment de mémoire.

## 4.1. Quand utiliser un index non cluster ?

Les index non-cluster sont utiles quand les utilisateurs doivent rechercher les données de différentes façons. Par exemple, il se peut qu'un utilisateur ait besoin de consulter régulièrement une base de données consacrée à la botanique pour trouver à la fois le nom commun et le nom scientifique des plantes. Vous pourriez créer un index non- cluster pour extraire les noms scientifiques et un index cluster pour récupérer les noms communs.

Les index non-cluster sont destinés à améliorer les performances des requêtes fréquemment utilisées qui ne sont pas couvertes par un index cluster. Si votre table possède déjà un index cluster et que vous devez indexer une autre colonne, vous n'avez pas d'autre solution que d'utiliser un index non-cluster. Avant de créer des index non- cluster, vous devez comprendre comment se déroule l'accès aux données.

Vous bénéficiez de performances de requête optimales lorsque l'index contient toutes les colonnes de la requête. Le terme *couverture* fait référence au nombre de colonnes impliquées dans une requête qui sont prises en charge par un index. Avec une couverture totale, l'optimiseur de requête peut rechercher toutes les valeurs de colonnes dans l'index; comme, de ce fait, vous n'accédez pas aux données du segment de mémoire ou de l'index cluster, il s'ensuit une diminution du nombre d'opérations d'E/S. Toutefois, les clés étendues et un nombre trop élevé d'index peuvent avoir un impact négatif sur les performances globales d'une base de données.

Envisagez l'utilisation d'un index non-cluster dans les cas suivants:

- Vous souhaitez améliorer les performances des requêtes utilisant les clauses JOIN ou GROUP BY. Créez alors plusieurs index non-cluster sur les colonnes impliquées dans les opérations de jointure et de regroupement, ainsi qu'un index cluster sur les colonnes de clé étrangère éventuelles.
- Votre table n'est que rarement mise à jour, mais elle contient d'importants volumes de données (une application d'aide à la décision, par exemple). De telles applications contiennent de grandes quantités de données qui sont principalement en lecture seule et peuvent tirer parti de l'existence de plusieurs index non-cluster. L'optimiseur de requête peut alors choisir au sein d'un plus grand nombre d'index afin de déterminer la méthode d'accès la plus rapide ; de plus, la faible fréquence de mise à jour de la base de données signifie qu'il est peu probable que la maintenance des index entrave les performances.
- Vos requêtes ne retournent pas des jeux de résultats volumineux.
- Vous devez indexer les colonnes qui figurent fréquemment dans les conditions de recherche d'une requête (clause WHERE, par exemple) et qui retournent des correspondances exactes.

Vous devez indexer les colonnes qui contiennent plusieurs valeurs distinctes, comme la combinaison du prénom et du nom. Les index non-cluster sont plus efficaces sur les colonnes dont la sélectivité des données s'étend de hautement sélectif à unique.

### 4.2. Considérations sur l'utilisation des index non cluster ?

Lorsque vous créez un index non-cluster, créez les index cluster avant les index non-cluster. SQL Server reconstruit automatiquement les index non-cluster existants dans les cas suivants :

- ✓ Suppression d'un index cluster existant
- ✓ Création d'un index cluster.
- ✓ Modification des colonnes définissant l'index cluster
- ✓ Évitez de définir un trop grand nombre d'index non-cluster si votre table est utilisée dans une application de traitement transactionnel en ligne (OLTP) ou si elle est fréquemment mise à jour. La définition de nombreux index sur une table affecte les performances des instructions INSERT, UPDATE et DELETE, car à mesure que les données de la table changent, tous les index doivent être mis à jour en conséquence

# Leçon 2 : Création des index

## 1. Vue d'ensemble de la création des index

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
   INDEX index_name ON { table | view } ( column [ ASC | DESC
   ] [ ,...n ] )
   INCLUDE  ( column  [ ,...n ] )
   [WITH option [ ,...n ] ]
   [ON {partition_scheme (column) |  filegroup | "default" } ]
```

| Option WITH | Objectif |
|---|---|
| ALLOW_ROW_LOCKS | Active/désactive les verrouillages des index au niveau ligne |
| ALLOW_PAGE_LOCKS | Active/désactive les verrouillages des index au niveau page |
| ONLINE | Active/désactive l'accès aux index pendant la création |
| FILLFACTOR | Contrôle l'espace libre sur les pages de niveau feuille |
| PAD_INDEX | Contrôle l'espace libre sur les pages de niveau non-feuille |

Vous pouvez créer un index en utilisant l'Explorateur d'objets de SQL Server Management Studio ou l'instruction Transact-SQL CREATE INDEX.
Pour créer un index avec Transact-SQL, utilisez l'instruction CREATE INDEX. L'instruction CREATE INDEX obéit à la syntaxe ci-dessus.
Les options WITH des instructions précédentes sont traitées dans la suite de la leçon.

L'exemple de code suivant crée un index non-cluster croissant intitulé **AK_Employee_LoginID** sur la colonne **LoginID** de la table **HumanResources.Employee** de la base de données **AdventureWorks**.

```
CREATE NONCLUSTERED INDEX [AK_Employee_LoginID]
ON [HumanResources].[Employee] ( [LoginID] ASC)
```

## 1.1. Options de verrouillage

ALLOW_ROW_LOCKS et ALLOW_PAGE_LOCKS sont de nouvelles options de SQL Server 2005. Le tableau suivant en récapitule les fonctions.

| Option | Description |
|---|---|
| ALLOW_ROW_LOCKS | Contrôle si les verrous au niveau de la ligne sont autorisés lors de l'accès à l'index. S'ils ne le sont pas, ce sont alors les verrous au niveau de la page et au niveau de la table qui sont utilisés. |
| ALLOW_PAGE_LOCKS | Contrôle si les verrous au niveau de la page sont autorisés lors de l'accès à l'index. S'ils ne le sont pas, ce sont alors les verrous au niveau de la ligne et au niveau de la table qui sont utilisés. |

Le moteur de base de données choisit le niveau approprié de verrouillage au sein des contraintes configurées à l'aide des options ALLOW_ROW_LOCKS et ALLOW_PAGE_LOCKS. Si les options ALLOW_ROW_LOCKS et ALLOW_PAGE_LOCKS ont toutes deux la valeur OFF, seuls les verrous au niveau de la table sont utilisés.

Voici l'exemple précédent avec, cette fois, les verrouillages de ligne non autorisés.

```
CREATE NONCLUSTERED INDEX [AK_Employee_LoginID] ON
[HumanResources].[Employee] ( [LoginID] ASC)
WITH ALLOW_ROW_LOCKS = OFF
```

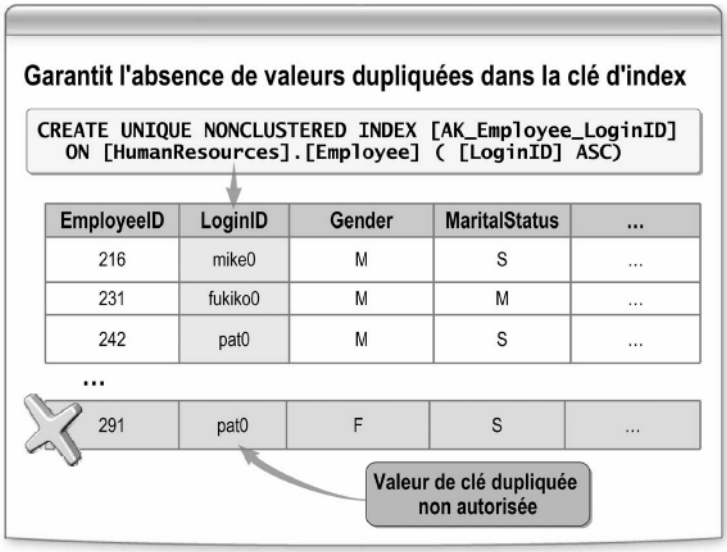
## 1.2. Travail en ligne et travail hors connexion

L'option ONLINE est également nouvelle dans SQL Server 2005. Elle spécifie si les tables sous-jacentes et les index associés sont disponibles pour les requêtes et les modifications de données pendant l'opération d'index associée à l'instruction CREATE INDEX.

Voici l'exemple précédent avec, cette fois, l'accès en ligne activé.

```
CREATE NONCLUSTERED INDEX [AK_Employee_LoginID] ON
[HumanResources].[Employee] ( [LoginID] ASC)
WITH ONLINE = ON
```

# 2. Qu'est-ce qu'un index unique ?




Un index unique est un index qui garantit que toutes les données d'une colonne indexée sont uniques et ne contiennent pas de valeurs dupliquées.

Quand existe un index unique, le moteur de base de données recherche les valeurs dupliquées chaque fois qu'une opération d'insertion ajoute des données. Les opérations d'insertion qui génèrent des valeurs de clé dupliquées sont annulées, et le moteur de base de données affiche un message d'erreur. Il en va de même si l'insertion modifie un grand nombre de lignes et ne génère qu'une seule valeur dupliquée. Cependant, si vous tentez d'entrer des données pour lesquelles il existe un index unique et que la clause IGNORE_DUP_KEY de l'instruction CREATE INDEX a la valeur ON, seules les lignes qui violent l'index unique échouent.

Quand vous créez un index sur une table qui contient déjà des données, le moteur de base de données s'assure qu'il n'existe pas de valeurs dupliquées.

### 2.1. Quand créer un index unique ?

Vous créez un index unique pour les index cluster ou non-cluster lorsque les données elles-mêmes sont uniques par essence. Créez uniquement des index uniques sur les colonnes dans lesquelles l'intégrité d'entité peut être appliquée. Par exemple, ne créez pas d'index unique sur la colonne **LastName** de la table **Person.Contact** de la base de données **AdventureWorks**, car il se peut que certains employés aient les mêmes noms. Si l'unicité doit être appliquée, créez les contraintes PRIMARY KEY ou UNIQUE sur la colonne au lieu de créer un index unique

**Remarque** Si une table possède une contrainte PRIMARY KEY ou UNIQUE,
SQL Server crée automatiquement un index unique lorsque vous exécutez l'instruction CREATE TABLE ou ALTER TABLE. Par défaut, il s'agit d'un index cluster, mais vous pouvez imposer que ce soit un index non-cluster avec l'option NONCLUSTERED de l'instruction CREATE TABLE.

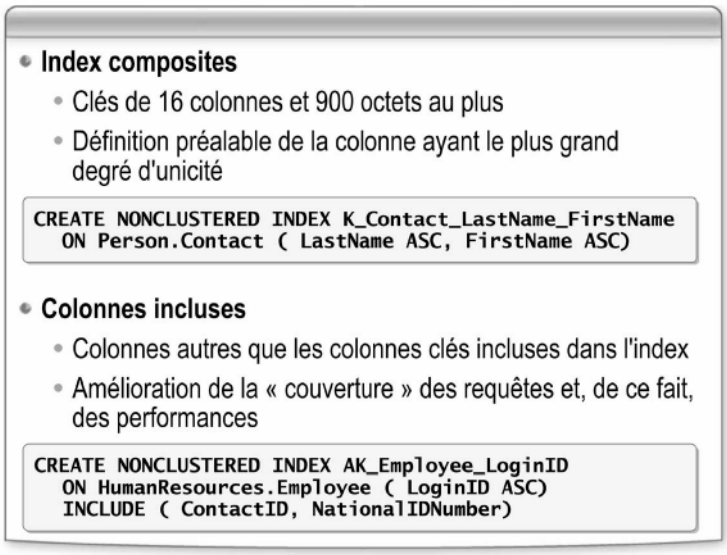
### 2.2. Exemple de création d'un index unique

Vous pouvez créer un index unique dans SQL Server Management Studio en activant la case à cocher Unique de la boîte de dialogue Nouvel index, lors de la création de l'index.

L'exemple suivant correspond au code Transact-SQL requis pour créer un index non- cluster croissant intitulé **AK_Employee_LoginID** sur la colonne **LoginID** de la table **HumanResources.Employee** de la base de données **AdventureWorks**.

```
CREATE UNIQUE NONCLUSTERED INDEX [AK_Employee_LoginID]
ON [HumanResources].[Employee] ( [LoginID] ASC)
```

# 3. Eléments à prendre en compte pour la création des index avec plusieurs colonnes




## 3.1. Qu'est-ce qu'un index composite ?

Un index composite spécifie plusieurs colonnes comme valeur de clé. L'utilisation d'index composites améliore les performances des requêtes, notamment quand les utilisateurs ont pour habitude de rechercher les informations de différentes façons. Cependant, les clés étendues augmentent les besoins de stockage d'un index.

Les index composites présentent les configurations et limitations suivantes:

- Vous pouvez combiner jusqu'à 16 colonnes pour former un même index composite.
- La somme des longueurs des colonnes de l'index composite ne peut pas dépasser 900 octets.
- La clause WHERE d'une requête doit faire référence à la première colonne de l'index composite pour que l'optimiseur de requête puisse utiliser celui-ci. Les index de couverture ne sont pas soumis à cette restriction.
- Les colonnes d'un index composite doivent toutes provenir de la même table, sauf quand un index est créé sur une vue, auquel cas elles doivent toutes provenir de la même vue.

Un index composite sur (colonne1, colonne2) n'est pas identique à un index sur (colonne2, colonne1), car chacun possède son propre ordre de colonnes.

## 3.2. Quand créer un index composite ?

Créez des index composites quand :

- Deux colonnes ou plus facilitent la recherche si elles forment une clé ;
- Les requêtes ne font référence qu'aux colonnes de l'index.

## 3.3. Conseils pour la création d'index composite ?

Lorsque vous créez un index composite, respectez les consignes suivantes:

- Définissez d'abord la colonne ayant le degré d'unicité le plus élevé. La première colonne de l'instruction CREATE INDEX est considérée comme ayant le rang le plus élevé.
- Utilisez les index composites pour les tables ayant plusieurs clés de colonne.
- Utilisez les index composites pour augmenter les performances des requêtes et réduire le nombre d'index que vous créez sur une table.

## 3.4. Colonnes incluses

SQL Server 2005 permet de spécifier des colonnes de table supplémentaires dont le contenu est stocké avec un index non-cluster. L'inclusion de colonnes non-clés permet de créer des index non-cluster qui couvrent un plus grand nombre de requêtes.

Les colonnes non-clés offrent les avantages supplémentaires suivants:

- Elles peuvent contenir des types de données qui ne sont pas autorisés dans les colonnes de clés d'index.

Elles ne sont pas prises en compte par le moteur de base de données lors du calcul du nombre de colonnes de clés d'index ou de la taille des clés d'index.

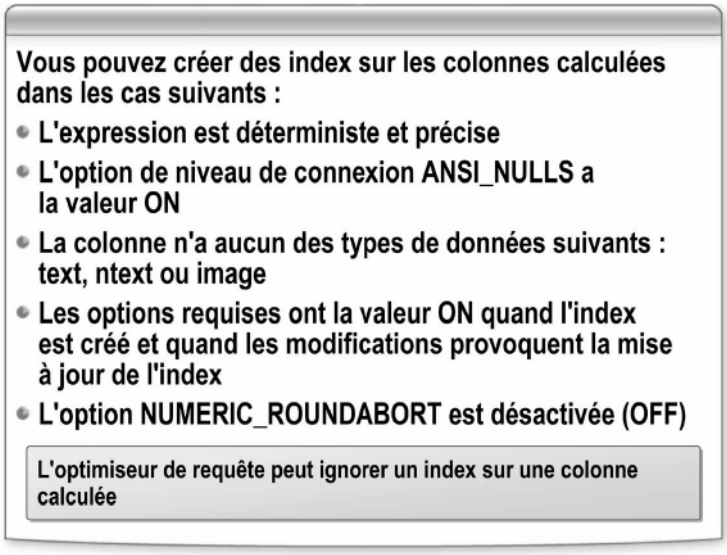
### 3.5. Exemple de création d'index composite

Vous pouvez créer un index ayant des colonnes incluses à l'aide de SQL Server Management Studio. À cette fin, sélectionnez les autres colonnes à inclure dans la section **Colonnes incluses** de la boîte de dialogue **Nouvel index**.

L'exemple suivant correspond au code Transact-SQL requis pour inclure les colonnes **ContactID** et **NationalIDNumber** d'un index intitulé **AK_Employee_LoginID** dans la colonne **LoginID** de la table **HumanResources.Employee** de la base de données **AdventureWorks**.

```
CREATE UNIQUE NONCLUSTERED INDEX [AK_Employee_LoginID]
ON [HumanResources].[Employee] ( [LoginID] ASC)
INCLUDE ( [ContactID], [NationalIDNumber])
```

## 4. Quand créer des index sur les colonnes calculées



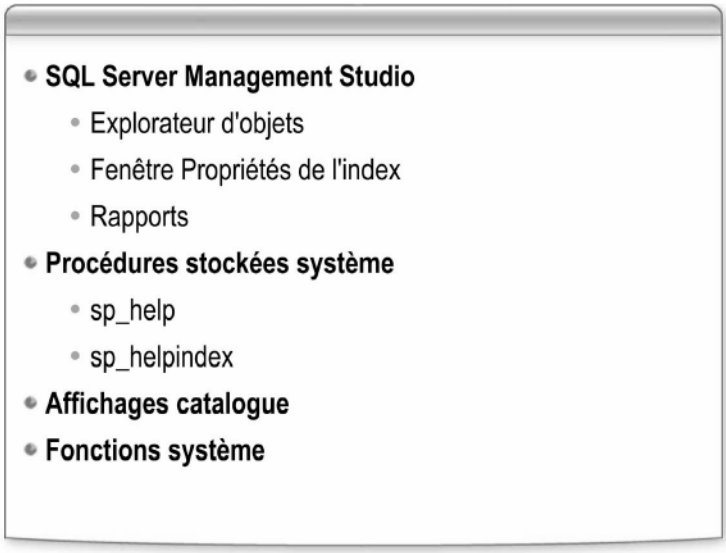

Vous pouvez créer des index sur les colonnes calculées dans les cas suivants:

- L'expression de colonne calculée est déterministe et précise. Les expressions déterministes retournent toujours le même résultat pour un ensemble donné d'entrées. Les expressions précises n'acceptent pas les types de données float et real et ne les incluent pas dans leur définition.
- L'option de niveau connexion ANSI_NULLS doit être activée (ON) lors de l'exécution de l'instruction CREATE TABLE. La fonction OBJECTPROPERTY indique si l'option est activée par le biais de la propriété IsAnsiNullsOn.
- L'expression de colonne calculée définie pour la colonne calculée ne peut pas avoir pour valeur les types de données text, ntext ou image.
- La connexion sur laquelle l'index est créé, ainsi que toutes les connexions tentant d'exécuter les instructions INSERT, UPDATE ou DELETE appelées à modifier des valeurs de l'index, doivent

comporter six options SET activées (ON) et une option désactivée (OFF). Les options suivantes doivent être activées (ON):

- ANSI_NULLS
- ANSI_PADDING?
- ANSI_WARNINGS
- CONCAT_NULL_YIELDS_NULL
- QUOTED_IDENTIFIER
- L'option NUMERIC_ROUNDABORT doit être désactivée (OFF).

# 5. Méthodes pour obtenir des informations sur les index




Il se peut que vous ayez besoin d'informations sur les index existants avant d'en créer, d'en modifier ou d'en supprimer un. SQL Server 2005 permet d'obtenir des informations sur les index de multiples façons. Celles-ci peuvent être classées en catégories de la manière suivante:

- SQL Server Management Studio
- Procédures stockées système
- Affichages catalogue
- Fonctions système

## 5.1. SQL Server Management Studio

SQL Server Management Studio propose des outils visuels pour afficher des informations sur les index au sein de l'environnement de gestion. Le tableau suivant répertorie les outils le plus souvent utilisés.

| | |
|---|---|
| **Explorateur d'objets** | Permet de se déplacer à travers la hiérarchie de la base de données pour consulter la liste des index d'une table |
| **Propriétés, fenêtre** | Permet d'afficher les propriétés d'un index. Pour y accéder, cliquez successivement sur l'index dans l'Explorateur d'objets, puis sur Propriétés |
| **Rapports** | Permet de créer les rapports de niveau base de données établissant comment les utilisateurs et le système emploient les index, ainsi que le nombre d'opérations exécutées sur les index. |

## 5.2.Procédures stockées

La procédure stockée **sp_helpindex** retourne des informations détaillées sur les index d'une table donnée. Pour chaque index, les informations retournées sont les suivantes: son nom, sa description et ses clés. La procédure stockée **sp_help** retourne des informations plus complètes pour les tables, mais elle inclut aussi les mêmes informations sur les index que celles retournées par **sp_helpindex**.

L'exemple suivant montre comment obtenir des informations sur les index de la table **Production.Product** avec la procédure stockée **sp_helpindex**.

```
EXEC sp_helpindex [Production.Product]
```

Le tableau suivant contient les résultats obtenus.

| index_name | index_description | index_keys |
|---|---|---|
| AK_Product_Name | non-cluster, unique, situé dans PRIMARY | Name |
| AK_Product_ProductNumber | non-cluster, unique, situé dans PRIMARY | ProductNumber |
| AK_Product_rowguid | non-cluster, unique, situé dans PRIMARY | rowguid |
| PK_Product_ProductID | cluster, unique, clé primaire située dans PRIMARY | ProductID |

## 5.3.Catalogue

Le tableau suivant répertorie les affichages catalogue qui fournissent des informations sur les index et le type d'informations qu'ils fournissent.

| Affichage catalogue | Informations proposées |
|---|---|
| sys.indexes | Type d'index, groupe de fichiers ou identificateur de schéma de la partition, et valeur en cours des options d'index stockées dans les métadonnées. |
| sys.index_columns | Identificateur de colonne, position dans l'index, type (clé ou non-clé) et ordre de tri (ASC ou DESC). |
| sys.stats | Statistiques associées à un index, y compris leur nom et l'information selon laquelle elles ont été créées automatiquement ou par l'utilisateur. |
| sys.stats_columns | Identificateur de colonne associé aux statistiques. |
| sys.xml_columns | Type d'index XML, primaire ou secondaire, type secondaire et description. |

## 5.4.Fonctions système

Le tableau suivant répertorie les fonctions système qui fournissent des informations sur les index, ainsi que le type d'informations proposé.

| Fonction | Informations proposées |
|---|---|
| sys.dm_db_index_physical_stats | Taille des index et statistiques de fragmentation |
| sys.dm_db_index_operational_stats | Index en cours et statistiques sur les E/S table |
| sys.dm_db_index_usage_stats | Statistiques d'utilisation des index par type de requête |
| INDEXKEY_PROPERTY | Position des colonnes d'index dans l'index et |

| | |
|---|---|
| | ordre de tri des colonnes (ASC ou DESC). |
| INDEXPROPERTY | Le type d'index, le nombre de niveaux et la valeur en cours des options d'index, stockés dans les métadonnées. |
| INDEX_COL | Nom de la colonne clé de l'index spécifié. |

# 6. Application pratique : création des index

## 6.1. Création d'index avec SQL Server Management studio

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, puis sur **Microsoft SQL Server 2005**, et cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Si l'Explorateur d'objets n'apparaît pas, cliquez sur **Explorateur d'objets** dans le menu **Affichage**.
4. Dans l'Explorateur d'objets, développez successivement **Bases de données**, **AdventureWorks**, **Tables**, **Person.Contact** et **Index**.
5. Cliquez avec le bouton droit sur **Index**, puis cliquez sur **Nouvel index**.
6. Dans la boîte de dialogue **Nouvel index**, entrez les valeurs suivantes:

| Propriété | Valeur |
|---|---|
| Nom de l'index | IX_Contact_LastName_FirstName |
| Type d'index | Non-cluster |

7. Cliquez sur le bouton **Ajouter**, sélectionnez les colonnes **FirstName** et **LastName**, puis cliquez sur **OK**.
8. Cliquez sur **LastName** dans la liste **Colonnes de clés d'index**, puis cliquez sur le bouton **Monter** pour déplacer **LastName** au-dessus de **FirstName**.
9. Cliquez sur la page **Options** située à gauche de la boîte de dialogue **Nouvel index**.
10. Activez la case à cocher **Définir le taux de remplissage**, puis entrez un taux de remplissage égal à **65**.
11. Activez la case à cocher **Autoriser le traitement en ligne des instructions DML lors de la création de l'index**.
12. Cliquez sur la page **Colonnes incluses** située à gauche de la boîte de dialogue **Nouvel index**.
13. Cliquez sur le bouton **Ajouter**, sélectionnez les colonnes **Title**,**MiddleName** et **Suffix**, puis cliquez sur **OK**.
14. Cliquez sur **OK** dans la boîte de dialogue **Nouvel index** pour créer l'index.
15. Une fois la commande terminée, cliquez avec le bouton droit sur le dossier **Index** de l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que l'index **IX_Contact_LastName_FirstName** a bien été créé.

## 4.1. Création d'index en Transact-SQL

Exécutez les étapes suivantes pour créer un index avec Transact-SQL:

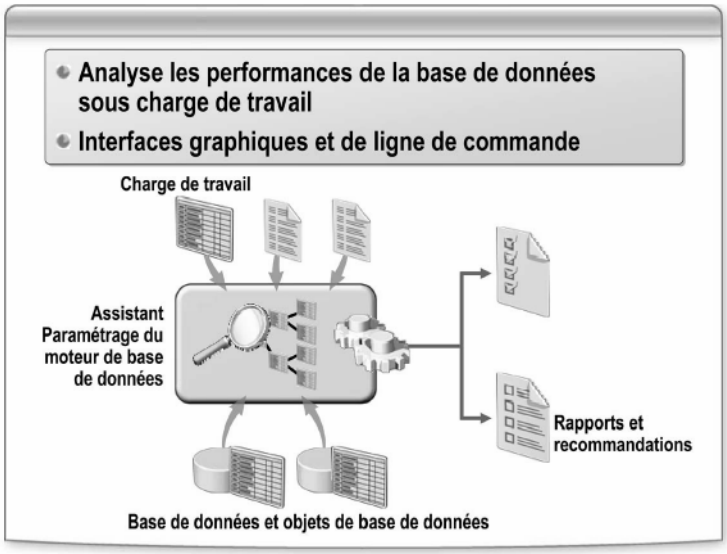
1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** de la barre d'outils.
2. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL ci-après.

```
USE AdventureWorks
CREATE NONCLUSTERED INDEX IX_Contact_LastName_FirstName_Transact
ON Person.Contact (LastName ASC,FirstName ASC)
INCLUDE (Title ,Middl eName , Suffix)
WITH (FILLFACTOR = 65, ONLINE = ON)
```

3. Cliquez sur le bouton **Exécuter** de la barre d'outils.
4. Une fois la commande terminée avec succès, cliquez avec le bouton droit sur le dossier **Index** de l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que l'index **IX_Contact_LastName_FirstName_Transact** a bien été créé.

# Leçon 3 : Optimisation des index

## 1. Qu'est-ce que l'Assistant Paramétrage du moteur de base de données ?




L'Assistant Paramétrage du moteur de base de données constitue un nouvel outil de SQL Server 2005. Avant SQL Server 2005, certaines fonctionnalités de cet Assistant étaient fournies par l'Assistant Paramétrage d'index. L'Assistant Paramétrage du moteur de base de données analyse un plus grand nombre de types d'événements et de structures, et fournit des recommandations de meilleure qualité.

L'Assistant Paramétrage du moteur de base de données propose deux interfaces:

- Un outil autonome doté d'une interface graphique utilisateur permettant de paramétrer les bases de données et de visualiser les recommandations et les rapports de paramétrage.
- Un utilitaire de ligne de commande, dta.exe, permettant d'accéder aux fonctionnalités de l'Assistant Paramétrage du moteur de base de données à partir des scripts et de la ligne de commande.

**Fonctionnement de l'assistant**
L'Assistant Paramétrage du moteur de base de données est un outil qui analyse l'impact des charges de travail sur les performances d'une ou plusieurs bases de données. Une charge de travail est un ensemble d'instructions Transact-SQL exécutées sur des bases de données à paramétrer. La source de la charge de travail peut être un fichier contenant des instructions Transact-SQL, un fichier trace généré par SQL Profiler, ou une table d'informations de trace également générée par SQL Profiler.

L'analyse peut être exécutée selon l'un ou l'autre des deux modes suivants.

| Mode d'analyse | Description |
|---|---|
| Évaluation | En mode évaluation, l'Assistant Paramétrage du moteur de base de données compare le coût de la configuration en cours (C) avec celui d'une configuration spécifiée par l'utilisateur (U), pour la même charge de travail. C est toujours une configuration réelle, parce qu'elle se compose de structures PDS présentes effectivement dans la base de données. Comparativement, U est une configuration composée de structures PDS physiques hypothétiques. Si l'Assistant Paramétrage du moteur de base de données indique que le coût de U est inférieur à celui de C, il est probable que la conception physique de U offre de meilleures performances que C. |
| Paramétrage | En mode paramétrage, un administrateur de base de données sait déjà qu'une partie de la conception physique de la base de données doit être corrigée, mais il souhaite que l'Assistant Paramétrage du moteur de base de données recommande les meilleures structures PDS pour le reste de la configuration. |

Après avoir analysé l'impact d'une charge de travail sur vos bases de données, l'Assistant Paramétrage du moteur de base de données fournit ses recommandations. Celles-ci incluent les modifications à apporter à la base de données, comme les index à créer ou à supprimer, et selon les options de paramétrage définies, des recommandations sur le partitionnement. Les recommandations sont fournies comme un ensemble d'instructions Transact-SQL qui effectueraient les modifications suggérées. Vous pouvez consulter le code Transact-SQL et l'enregistrer en vue d'une révision ou application ultérieure, ou choisir d'implémenter immédiatement les modifications recommandées.

# 2. Application pratique : utilisation de l'Assistant Paramétrage du moteur de base de données

## 2.1. Analyser une base de données avec l'Assistant Paramétrage du moteur de base de données

Exécutez les étapes suivantes pour utiliser l'Assistant Paramétrage du moteur de base de données afin d'analyser une base de données:

1. Cliquez sur **Démarrer**, pointez successivement sur **Tous les programmes**, **Microsoft SQL Server 2005** et **Outils de performances**, puis cliquez sur **Assistant Paramétrage du moteur de base de données**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Dans la zone **Nom de session**, tapez **Module4**.
4. Sous **Charge de travail**, vérifiez que **Fichier** est sélectionné, puis cliquez sur le bouton **Rechercher un fichier de charge de travail**.
5. Naviguez jusqu'au dossier **D:\Practices.**
6. Cliquez sur **Workload.sql**, puis sur **Ouvrir**.
7. Sélectionnez **AdventureWorks** dans la liste **Base de données pour l'analyse de la charge de travail**.
8. Dans la section **Sélectionnez les bases de données et tables à analyser**, sélectionnez la base de données **AdventureWorks**.
9. Cliquez sur la flèche de déroulement de la colonne **Tables sélectionnées** de la base de données **AdventureWorks**.
10. Désactivez la case à cocher de l'en-tête de colonne **Nom** pour effacer toutes les tables, puis sélectionnez l'une après l'autre les tables **SalesOrderDetail** et **SalesOrderHeader**.

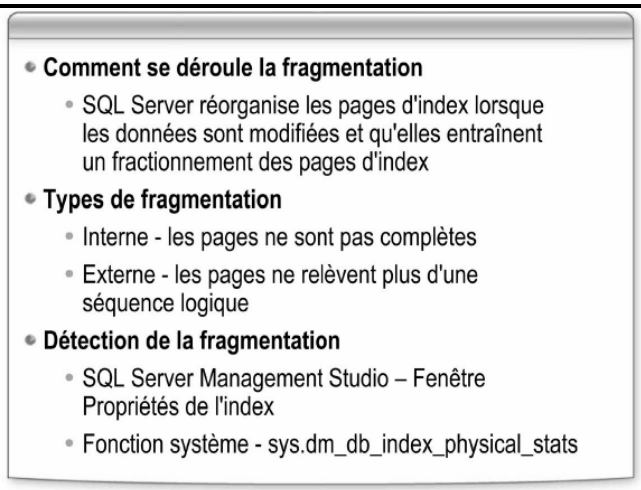
11. Cliquez à nouveau sur la flèche de déroulement de la colonne **Tables sélectionnées** de la base de données **AdventureWorks** pour masquer la liste des tables.
12. Cliquez sur l'onglet **Options de paramétrage**.
13. Désactivez la case à cocher **Limiter la durée du paramétrage**.
14. Sélectionnez **Ne pas conserver de structures PDS existantes**.
15. Cliquez sur **Options avancées**.
16. Sélectionnez **Générer des recommandations en ligne dès que possible**.
17. Cliquez sur **OK** pour fermer la boîte de dialogue **Options de paramétrage avancées**.
18. Dans le menu **Actions**, cliquez sur **Démarrer l'analyse**, puis attendez que l'analyse soit terminée.

## 2.2. Vérifier les recommandations de l'Assistant Paramétrage du moteur de base de données

Exécutez les étapes suivantes pour examiner les recommandations faites par l'Assistant Paramétrage du moteur de base de données:

1. Dans la fenêtre **Recommandations**, vérifiez que la case à cocher **Afficher les objets existants**, située sous la liste des index, est bien désactivée.
2. Cliquez sur l'onglet **Rapports**.
3. Examinez les informations du volet **Résumé de paramétrage** et notez la valeur **Pourcentage d'amélioration attendu**.
4. Dans la liste **Sélectionnez un rapport**, cliquez sur **Rapport d'utilisation d'index (en cours)**.
5. Faites défiler la fenêtre vers la droite jusqu'à ce qu'apparaisse la colonne **Taux d'utilisation**.
6. Dans la liste **Sélectionnez un rapport**, cliquez sur **Rapport d'utilisation d'index (recommandé)**.
7. Faites défiler la fenêtre vers la droite jusqu'à ce qu'apparaisse la colonne **Taux d'utilisation**. Notez les différences entre ce rapport et le précédent.
8. Cliquez sur l'onglet **Recommandations**.
9. Faites défiler la fenêtre vers la droite jusqu'à ce qu'apparaisse la colonne **Définition**.
10. Cliquez sur la première définition, puis examinez le code Transact-SQL. Notez la présence du bouton **Copier dans le Presse-papiers**.
11. Cliquez sur **Terminer** pour fermer la fenêtre Aperçu de script SQL.

# 3. Fragmentation des index




La fragmentation des index constitue une utilisation inefficace des pages au sein d'un index. La fragmentation se produit au fur et à mesure de la modification des données. Par exemple, lors de l'ajout de lignes de données à une table ou de leur suppression, ou lors de la modification des valeurs des colonnes indexées, SQL Server ajuste les pages d'index de façon à accueillir les modifications et à maintenir le stockage des données indexées. L'ajustement des pages d'index est appelé *fractionnement de page*. Le fractionnement augmente la taille d'une table et le temps nécessaire au traitement des requêtes.

## 3.1. Types de fragmentation

Comme indiqué dans le tableau suivant, il existe deux types de fragmentation d'index: la fragmentation externe et la fragmentation interne.

| Type de fragmentation | Description |
|---|---|
| Interne | Utilisation inefficace des pages dans un index, car la quantité de données stockée dans chaque page est inférieure à ce que la page de données peut contenir. La fragmentation interne se traduit par une augmentation des E/S logiques et physiques, et une mémoire plus importante est nécessaire pour stocker les lignes dans le cache. Ces lectures supplémentaires peuvent conduire à une dégradation des performances des requêtes. Toutefois, les tables à insertions intensives peuvent tirer parti de la fragmentation interne, car la probabilité qu'une insertion provoque un fractionnement de page est moins élevée. |
| Externe | Utilisation inefficace des pages dans un index, car l'ordre logique des pages n'est pas correct. La fragmentation externe ralentit l'accès disque aux lignes en raison de la recherche des têtes de disque et n'est jamais bénéfique. |

## 3.2. Détection de la fragmentation

Vous pouvez utiliser SQL Server Management Studio ou la fonction de gestion dynamique **sys.dm_db_index_physical_stats** pour déterminer l'étendue de la fragmentation de vos index.

Pour afficher les détails de la fragmentation d'un index dans SQL Server Management Studio, ouvrez la fenêtre Propriétés de l'index correspondant, puis cliquez sur la page **Fragmentation**. En plus de propriétés générales sur les pages de l'index, la fenêtre Propriétés affiche le remplissage moyen de pages pleines et la fragmentation totale de l'index sous forme de pourcentages. Plus la valeur est élevée, plus l'index est fragmenté.

Vous pouvez utiliser **sys.dm_db_index_physical_stats** pour afficher la fragmentation d'un index spécifique, de tous les index d'une table ou d'une vue indexée, de tous les index d'une base de données ou de tous les index de toutes les bases de données. Les arguments transmis à la fonction **sys.dm_db_index_physical_stats** incluent les ID de la base de données, table, index et partition que vous

souhaitez analyser. L'ensemble des résultats de la fonction inclut la colonne **avg_fragmentation_in_percent**, qui affiche la fragmentation moyenne des index sous forme de pourcentage (même illustration que celle de la fenêtre Propriétés de l'index dans SQL Server Management Studio).

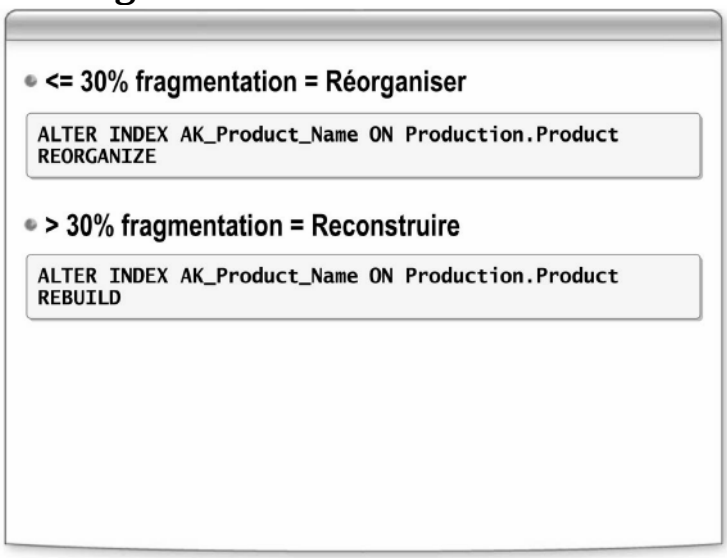
L'exemple de code suivant montre comment obtenir la fragmentation moyenne de tous les index de la table **Production.Product** à l'aide de **sys.dm_db_index_physical_stats**.

```
SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID('AdventureWorks'),
     OBJECT_ID('Production.Product'), NULL, NULL, NULL) AS a
     JOIN sys.indexes AS b ON a.object_id = b.object_id AND a.index_id = b.index_id;
```

Les résultats doivent se présenter de la façon suivante:

| index_id | name | avg_fragmentation_in_percent |
|---|---|---|
| 1 | PK_Product_ProductID | 23.0769230769231 |
| 2 | AK_Product_ProductNumber | 50 |
| 3 | AK_Product_Name | 66.6666666666667 |
| 4 | AK_Product_rowguid | 50 |

# 4. Options de défragmentation des index




Il existe deux options pour défragmenter un index : réorganisation et reconstruction.

La réorganisation d'un index défragmente le niveau feuille des index cluster et non- cluster des tables : les pages de niveau feuille sont réorganisées physiquement afin de correspondre à l'ordre logique (de gauche à droite) des nœuds feuille. Les performances de l'analyse des index sont meilleures lorsque les pages sont ordonnées. L'index est réorganisé dans les pages existantes qui lui sont allouées ; aucune nouvelle page n'est
allouée. Si un index recouvre plusieurs fichiers, ceux-ci sont réorganisés l'un après l'autre. Les pages ne migrent pas d'un fichier à l'autre.

La réorganisation entraîne aussi une compression des pages d'index. Toutes les pages vides créées par cette compression sont supprimées, ce qui permet de libérer de l'espace disque. La compression est basée sur la valeur du taux de remplissage de l'affichage catalogue **sys.indexes**.

La reconstruction d'un index implique la suppression de l'index et la création d'un nouvel index. Ainsi, la fragmentation est supprimée, l'espace disque est plus important grâce à la compression des pages à l'aide de la valeur du taux de remplissage spécifié ou

existant, et les lignes d'index sont réorganisées en pages contiguës (en allouant de nouvelles pages en fonction des besoins). Ceci peut améliorer les performances du disque, car le nombre de lectures de pages nécessaire pour obtenir les données demandées est moins élevé.

### 4.1. Différences entre réorganisation et reconstruction d'index

Le choix entre la réorganisation ou la reconstruction d'un index pour supprimer la fragmentation dépend du niveau de fragmentation de l'index, tel qu'il est mentionné par SQL Server Management Studio ou **sys.dm_db_index_physical_stats**. Le tableau suivant fournit des conseils sur la meilleure approche à suivre en matière de suppression des différents degrés de fragmentation

| avg_fragmentation_in_percent | Action |
|---|---|
| < = 30% | Réorganiser |
| > 30% | Reconstruire |

### 4.2. Regénération d'un index

Vous pouvez utiliser la clause REORGANIZE de l'instruction ALTER INDEX pour défragmenter les index.

L'instruction ALTER INDEX avec la clause REORGANIZE remplace l'instruction DBCC INDEXDEFRAG des versions antérieures de SQL Server.

L'exemple de code suivant illustre la syntaxe de l'instruction ALTER INDEX lorsqu'elle est utilisée pour réorganiser l'index **AK_Product_Name** de la table **Production.Product**.

```
ALTER INDEX AK_Product_Name
ON Production.Product
REORGANIZE
```

Vous pouvez également réorganiser *tous* les index d'une table, comme illustré dans l'exemple ci-après.

```
ALTER INDEX ALL ON Production.Product
REORGANIZE
```

### 4.3. Reconstruction d'un index

Vous pouvez utiliser la clause REINDEX de l'instruction ALTER INDEX pour défragmenter les index. L'instruction ALTER INDEX avec la clause REBUILD remplace l'instruction DBCC DBREINDEX des versions antérieures de SQL Server.

L'exemple de code suivant illustre la syntaxe de l'instruction ALTER INDEX lorsqu'elle est utilisée pour reconstruire l'index **AK_Product_Name** de la table **Production.Product**.

```
ALTER INDEX AK_Product_Name ON Production.Product REBUILD
```

## 5. Application pratique : Défragmentation des index

### 5.1. Identifier la fragmentation d'un index

Exécutez les étapes suivantes pour identifier les index de la table **HumanResources.Employee** qui contiennent une fragmentation :

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, puis sur **Microsoft SQL Server 2005**, et cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur,** puis cliquez sur **Se connecter**.
3. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** de la barre d'outils.

4. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL ci-après.

```
USE AdventureWorks
SELECT a.index_id as [Index ID], name as [Index Name], avg_fragmentation_in_percent as
Fragmentation
FROM sys.dm_db_index_physical_stats (DB_ID('AdventureWorks'),
          OBJECT_ID('HumanResources.Employee'),NULL, NULL, NULL) AS a
          JOIN sys.indexes AS b ON a.object_id = b.object_id
          AND a.index_id = b.index_id
ORDER BY Fragmentation DESC
```

5. Cliquez sur le bouton **Exécuter** de la barre d'outils.
6. Lorsque la commande est terminée, examinez les résultats et identifiez un index qui contient un niveau élevé de fragmentation, par exemple l'index **PK_Employee_EmployeeID**.

## 5.2. Pour défragmenter un index

Exécutez les étapes suivantes pour défragmenter l'index **PK_Employee_EmployeeID**:

1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** de la barre d'outils.
2. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL ci-après.

```
USE AdventureWorks
ALTER INDEX PK_Employee_EmployeeID ON HumanResources. Employee REBUILD
```

3. Cliquez sur le bouton **Exécuter** de la barre d'outils.
4. Lorsque la commande est terminée, réexécutez la requête précédente pour obtenir les niveaux de fragmentation et examiner les résultats modifiés.

Fermez SQL Server Management Studio. Cliquez sur **Non** si vous êtes invité à enregistrer les fichiers.

# 6. Atelier pratique : Création des index

## 6.1. Scénario

Le service Production d'Adventure Works a consacré les six derniers mois à contrôler ses procédures et ses besoins en termes de prise en charge des applications. Cette révision a permis de mettre en évidence différents problèmes nécessitant une attention immédiate et un certain nombre d'initiatives nouvelles. Au démarrage de chaque initiative, son impact sur la base de données et les nouvelles exigences qu'elle induit sont débattus avec le développeur de base de données senior afin de convenir des actions appropriées à prendre. Voici le premier ensemble de tâches que le développeur de base de données senior vous a demandé d'effectuer:

- Vous devez utiliser SQL Server Management Studio pour créer un projet de scripts SQL Server relatifs aux modifications dans le dossier D:\Labfiles\Starter.
- Vous devez implémenter un nouvel index unique non-cluster intitulé Ix_Product_Supply_Chain sur la table Production.Product pour prendre en charge les requêtes qui incluent des informations sur la gestion des stocks. Utilisez les paramètres suivants:
  - *Colonnes clés*: **ProductNumber**, **Color**, **ReorderPoint**, **SafetyStockLevel**.
  - *Colonnes incluses*: **DaysToManufacture**.
  - *Verrouillage*: autorisez le verrouillage des lignes, mais pas des pages.
  - *Taux de remplissage*: définissez le taux de remplissage à la valeur 90 % sur les nœuds feuille et non-feuille.

Les analystes qui travaillent pour le service Production ont identifié les requêtes qui sont le plus souvent appelées par leurs applications. Ces requêtes sont fournies dans un fichier intitulé Queries.sql, situé dans le dossier D:\Labfiles\Starter, et reproduites ci-dessous. Vous devez évaluer les performances de ces requêtes en fonction de la configuration d'index en cours et fournir des

recommandations sur les améliorations qui doivent être apportées.

```
SELECT *
FROM Production.
Product ORDER BY
Name ASC

SELECT Name, ProductNumber, ListPrice AS Price
FROM Production.Product
WHERE ProductLine = 'R' AND DaysToManufacture < 4 ORDER
BY Name ASC

SELECT p.Name AS ProductName,
       NonDiscountSales = (OrderQty * UnitPrice),
       Discounts = ((OrderQty * UnitPrice) * UnitPriceDiscount)
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail sod ON p.ProductID = sod.ProductID
ORDER BY ProductName DESC

SELECT 'Total income is',
       ((OrderQty * UnitPrice) * (1.0 - UnitPriceDiscount)),
       ' for ',
       p.Name AS ProductName
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail sod
ON p.ProductID = sod.ProductID
ORDER BY ProductName ASC
```

**Autres informations**

Lorsque vous développez des bases de données, il peut être utile de recourir à SQL Server Management Studio pour créer un projet de scripts SQL Server et l'utiliser pour documenter le code Transact-SQL nécessaire, le cas échéant, à la recréation de la solution.

Utilisez la procédure suivante pour créer un projet de scripts SQL Server:

1. Ouvrez SQL Server Management Studio, en vous connectant au serveur que vous souhaitez gérer.
2. Dans le menu **Fichier**, pointez sur **Nouveau**, puis cliquez sur **Projet.**
3. Sélectionnez le modèle **Scripts SQL Server** et entrez un nom approprié et un emplacement pour le projet. Notez que vous pouvez créer une solution qui contient plusieurs projets, mais dans de nombreux cas, un seul projet par solution convient parfaitement.

Pour ajouter un fichier de requête à un projet :

1. Cliquez sur **Nouvelle requête** dans le menu **Projet** ou cliquez avec le bouton droit sur le dossier **Requêtes** de l'Explorateur de solutions, puis cliquez sur **Nouvelle requête**. Si l'Explorateur de solutions n'apparaît pas, vous pouvez l'afficher en cliquant sur **Explorateur de solutions** dans le menu **Affichage**.
2. Lorsque vous y êtes invité, connectez-vous au serveur sur lequel vous voulez stocker la requête. Un objet de connexion est ajouté au projet.
3. Modifiez le nom par défaut du fichier de requête (**SQLQuery1.sql**) en cliquant dessus avec le bouton droit dans l'Explorateur de solutions, puis en sélectionnant **Renommer**.

Bien que vous puissiez effectuer toutes les tâches de développement de la base de données en exécutant des instructions Transact-SQL, il est souvent plus facile d'utiliser l'interface graphique utilisateur de SQL Server Management Studio. Cependant, vous devez générer les scripts Transact-SQL correspondants et les enregistrer dans le projet à des fins de consultation ultérieure.

Souvent, vous pouvez générer le script Transact-SQL d'une action avant de cliquer sur **OK** dans la boîte de dialogue **Propriétés** utilisée pour exécuter l'action. Plusieurs boîtes de dialogue **Propriétés** incluent une liste déroulante **Script** avec laquelle vous pouvez générer une action de script vers une nouvelle fenêtre de

requête, un fichier, le Presse- papiers ou un travail de SQL Server Agent. Une technique courante consiste à ajouter un fichier de requête vide à un projet, puis à écrire chaque action de script dans le Presse- papiers au fur et à mesure qu'elles sont exécutées et à coller le script généré dans le fichier de requête.

Vous pouvez également générer des scripts pour plusieurs objets existants, comme les bases de données et les tables. Pour générer un script, cliquez avec le bouton droit sur l'objet dans l'Explorateur d'objets et écrivez le script de l'action CREATE. Si l'Explorateur d'objets n'apparaît pas, vous pouvez l'afficher en cliquant sur **Explorateur d'objets** dans le menu **Affichage**.

## 6.2.Exercice 1 : Création des index

**Création de l'index Ix_Product_Supply_Chain**

| Tâche | Informations |
|---|---|
| Créer un projet de scripts SQL Server. | 1. Démarrez SQL Server Management Studio et connectez-vous au serveur **MIAMI** à l'aide de l'authentification Windows.<br>2. Créez un nouveau projet de scripts SQL Server intitulé **AW_Indexes**. |
| Créer l'index **Ix_Product_Supply_Chain**. | 1. Ajoutez un nouveau fichier de requête au projet. Lorsque vous y êtes invité, connectez-vous au serveur **MIAMI** en utilisant l'authentification Windows.<br>2. Renommez **SQLQuery1.sql** en **CreateIndex.sql**.<br>3. Dans la fenêtre de requête, tapez l'instruction Transact-SQL appropriée pour créer l'index **Ix_Product_Supply_Chain** de la table **Production.Products** de la base de données **AdventureWorks**.<br>4. Exécutez la requête, puis enregistrez le fichier de requête.<br>5. Utilisez l'Explorateur d'objets pour vérifier que l'index a été créé.<br>6. Ne fermez pas SQL Server Management Studio ; vous allez l'utiliser dans l'exercice suivant. |

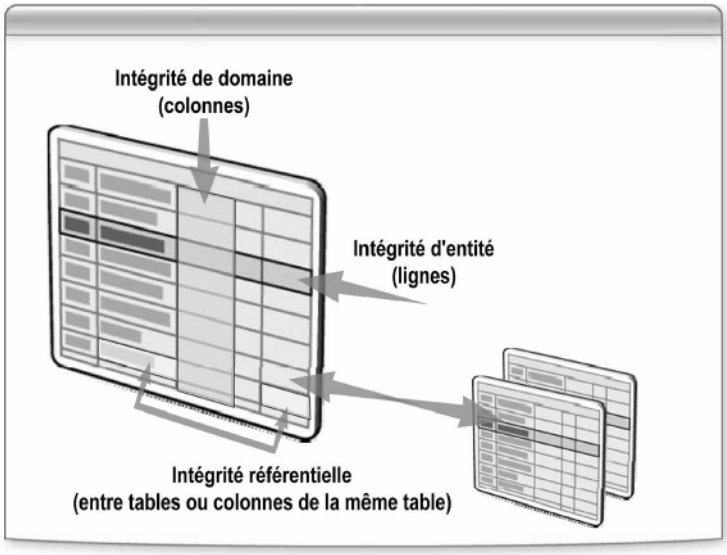
## 6.3.Exercice 2 : Paramétrage des index

| Tâche | Informations |
|---|---|
| Ajouter la requête de charge de travail au projet. | 1. Dans le menu **Projet**, cliquez sur **Ajouter un élément existant** et ajoutez le fichier Queries.sql au dossier D:\Labfiles\Starter. Lorsque vous y êtes invité, connectez-vous au serveur **MIAMI**.<br>2. Examinez les requêtes du fichier. Vous allez utiliser ces requêtes pour analyser les index de la base de données **AdventureWorks**. |
| Analyser l'utilisation des index dans la base de données **AdventureWorks**. | 1. Ouvrez l'Assistant Paramétrage du moteur de base de données et connectez-vous au serveur **MIAMI** en utilisant l'authentification Windows. Renommez la nouvelle session par défaut en **AW_TuneIndexes**.<br>2. Configurez la session pour charger la charge de travail à partir du fichier **Queries.sql**.<br>3. Choisissez uniquement les tables **Product** et **SalesOrderDetail** pour l'analyse de la base de données **AdventureWorks**.<br>4. Configurez une session sans date d'expiration, autorisée à utiliser les index, mais pas les partitions, et dans laquelle la totalité de la structure de la base de données existante est conservée.<br>5. Exécutez l'analyse, puis recherchez des recommandations pour modifier les index existants ou en créer de nouveaux.<br>6. Dans le projet **AW_Indexes** de SQL Server Management Studio, créez une nouvelle requête, intitulée **IndexRecommendations.sql**.<br>7. Pour toutes les recommandations de l'Assistant Paramétrage du moteur de base de données qui créent ou modifient des index, copiez les instructions Transact-SQL et collez-les dans **IndexRecommendations.sql**.<br>8. Exportez les résultats de la session de l'Assistant Paramétrage du moteur de base de données vers un fichier intitulé **ProductionAnalysis.xml** en cliquant sur **Fichier**, puis sur **Exporter les résultats de la session**.<br>9. Fermez l'Assistant Paramétrage du moteur de base de données, mais pas SQL Server Management Studio. Vous allez l'utiliser dans le prochain exercice. |

# MODULE 4 : IMPLEMENTATION DE L'INTEGRITE DES DONNEES

## Leçon 1 : Présentation de l'intégrité des données

### 1. Type d'intégrité des données




L'application de l'intégrité des données garantit la qualité des données stockées dans la base de données. Les divers types d'intégrité des données que vous devez prévoir sont les suivants :

- Intégrité de domaine (ou de colonne)
- Intégrité d'entité
- Intégrité référentielle

**Intégrité de domaine**
L'intégrité de domaine (ou de colonne) permet de spécifier un ensemble de valeurs de données valides pour une colonne et de déterminer s'il faut autoriser des valeurs NULL. L'intégrité de domaine est souvent appliquée par un contrôle de validité et peut l'être en restreignant le type de données, le format ou la plage de valeurs possibles autorisées dans une colonne.

**Intégrité d'entité**
L'intégrité d'entité (ou de table) exige que toutes les lignes d'une table disposentd'un identificateur unique connu en tant que valeur de clé primaire. La possibilité de modifier la valeur de clé primaire ou de supprimer la ligne tout entière dépend du niveau d'intégrité requis entre la clé primaire et toutes les autres tables.

**Intégrité référentielle**
L'intégrité référentielle garantit que les relations entre les clés primaires (dans la table référencée) et les clés étrangères (dans les tables de référence) sont toujours conservées. À moins que l'action en cascade soit permise, vous ne pouvez pas supprimer une ligne dans une table référencée ni modifier la clé primaire si une clé étrangère fait référence à la ligne. Vous pouvez définir des relations d'intégrité référentielle au sein de la même table ou entre des tables distinctes.

### 2. Options de mise en application de l'intégrité des données

| Mécanisme | Description |
|---|---|
| Types de données | Définir le type des données pouvant être stockées dans une colonne |
| Règles | Définir les valeurs acceptables pouvant être insérées dans une colonne |
| Valeurs par défaut | Définir la valeur d'une colonne si aucune valeur n'est spécifiée |
| Contraintes | Définir la manière dont le moteur de base de données applique l'intégrité des données |
| Déclencheurs | Définir le code exécuté automatiquement en cas de modification d'une table |
| Schémas XML | Définir le contenu acceptable de documents XML et de fragments insérés dans une colonne de données **xml** |

Le tableau ci-dessous résume les mécanismes fournis par SQL Server 2005 pour la mise en application de l'intégrité des données.

| Mécanisme | Types d'intégrité | Description |
|---|---|---|
| Types de données | Domaine | Les types de données permettent d'appliquer des restrictions fondamentales sur le type de données qu'il est possible de stocker dans chaque colonne. L'affectation de types de données à chaque colonne dans une table est la première étape à suivre dans l'application de l'intégrité des données. Toutefois, les types de données seuls ne suffisent pas pour appliquer le type d'intégrité des données requis dans des solutions commerciales sans les autres mécanismes répertoriés dans ce tableau. Par exemple, une colonne déclarée de type **int** garantit que seules les valeurs entières numériques peuvent être entrées mais ne peut d'elle-même limiter les valeurs à la plage 0 à 1000. |
| Règles et valeurs par défaut | Domaine, entité | Les règles définissent les valeurs acceptables que peut contenir une colonne ; les valeurs par défaut permettent de définir la valeur d'une colonne si aucune valeur n'est spécifiée. Le plus important à retenir est que les règles et les valeurs par défaut sont des objets indépendants que vous pouvez lier à une ou plusieurs colonnes ou types de données définis par l'utilisateur, ce qui permet de les définir une seule fois et de les utiliser à plusieurs reprises. Leur utilisation présente néanmoins un inconvénient puisqu'elles ne sont pas conformes à la norme ANSI (American National Standards Institute). Vous devez utiliser des contraintes au lieu de règles et de valeurs par défaut. |

| | | |
|---|---|---|
| Contraintes | Domaine, entité, référentiel | Les contraintes permettent de définir la manière dont le moteur de base de données SQL Server 2005 applique automatiquement l'intégrité d'une base de données. Elles définissent des restrictions pour les valeurs autorisées dans les colonnes et constituent le mécanisme standard de mise en application de l'intégrité. L'utilisation de contraintes est préférable à l'utilisation de déclencheurs, de règles et de valeurs par défaut. L'optimiseur de requête utilise également des définitions de contraintes pour créer des plans d'exécution de requêtes très performants. |
| Déclencheurs | Domaine, entité, référentiel | Les déclencheurs sont une forme spéciale de procédure stockée qui s'exécute lorsqu'un événement de langage de manipulation de données (DML, Data Manipulation Language) se produit sur le serveur de base de données. Les événements DML comprennent des instructions UPDATE, INSERT ou DELETE émises par rapport à une table ou à une vue. Les déclencheurs sont employés pour appliquer des règles métier en cas de modification des données. |
| schémas XML | Domaine (XML) | Les schémas XML permettent de définir l'espace de noms, la structure et le contenu acceptable des documents XML. En appliquant un schéma XML à une colonne de table définie avec le type de données **xml**, vous pouvez restreindre la structure et le contenu des données XML stockées dans cette colonne. |

# Leçon 2 : Implémentation de contraintes

## 1. Que sont les contraintes ?

| Type d'intégrité | Type de contrainte | Description |
|---|---|---|
| Domaine | DEFAULT | Spécifie la valeur par défaut de la colonne |
| | CHECK | Spécifie la valeur autorisée de la colonne |
| | FOREIGN KEY | Spécifie la colonne dans laquelle les valeurs doivent exister |
| | NULL | Spécifie si la valeur NULL est autorisée |
| Entité | PRIMARY KEY | Identifie chaque ligne de manière unique |
| | UNIQUE | Empêche la duplication de clé non primaires |
| Référentiel | FOREIGN KEY | Définit des colonnes dont la valeur doit correspondre à la clé primaire de cette table |
| | CHECK | Spécifie la valeur autorisée pour une colonne sur la base du contenu d'une autre colonne |

Les contraintes forment une méthode de mise en application de l'intégrité des données reposant sur la norme ANSI. Chaque type d'intégrité de données de domaine, d'entité ou de référentiel est appliqué à l'aide de types de contraintes distincts. Les contraintes garantissent que les valeurs de données valides sont entrées dans des colonnes et que les relations entre les tables sont maintenues. Le tableau ci-dessous décrit les divers types de contraintes disponibles dans SQL Server 2005 et indique s'ils sont définis dans une table ou une colonne.

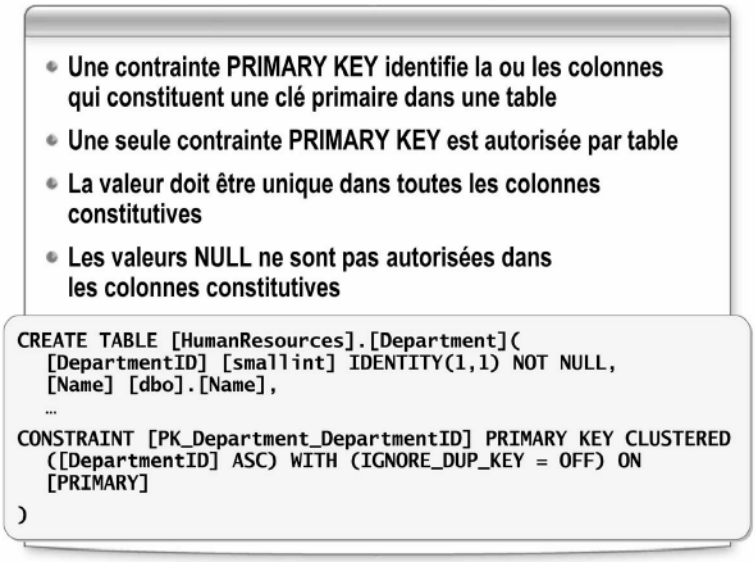
### 1.1. Création de contraintes

Vous pouvez créer des contraintes lorsque vous créez une table à l'aide de l'instruction CREATE TABLE. Une contrainte au niveau de la colonne s'applique à une seule colonne et est inhérente à la définition de la colonne. Une contrainte au niveau de la table peut référencer une ou plusieurs colonnes de la table. Les contraintes au niveau de la table sont spécifiées ensemble dans une section séparée une fois que toutes les colonnes ont été définies. Vous pouvez modifier les contraintes d'une table existante à l'aide de l'instruction ALTER TABLE. Vous pouvez ajouter des contraintes à une table avec des données existantes et insérer des contraintes dans une seule ou plusieurs colonnes.

Le code ci-dessous est une représentation simplifiée de la syntaxe de l'instruction CREATE TABLE qui indique où les diverses contraintes au niveau de la colonne et de la table sont à déclarer.

```
CREATE TABLE table_name
  ({ < column_definition > | < table_constraint > } [ ,...n ])
< column_definition > ::=
  {column_name data_type }
  [{DEFAULT constant_expression | [ IDENTITY [( seed , increment )]]}]
  [ < column_constraint > [ ...n ] ]
< column_constraint > ::=
  [ CONSTRAINT constraint_name ]
  { [ NULL | NOT NULL ]
    | [ PRIMARY KEY | UNIQUE ]
    | REFERENCES ref_table [ ( ref_column ) ]
    [ ON DELETE { CASCADE | NO ACTION } ]
    [ ON UPDATE { CASCADE | NO ACTION } ]
  }
< table_constraint > ::=
  [ CONSTRAINT constraint_name ]
  { [ { PRIMARY KEY | UNIQUE } { (column [ ,...n ] ) } ]
  | FOREIGN KEY (column [ ,...n ] )
     REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
     [ ON DELETE { CASCADE | NO ACTION } ]
     [ ON UPDATE { CASCADE | NO ACTION } ]
  }
```

## 2. Contraintes PRIMARY KEY




Une contrainte PRIMARY KEY définit la ou les colonnes qui constituent une clé primaire dans une table. La clé primaire identifie de manière unique une ligne dans une table et permet d'appliquer l'intégrité d'entité de cette dernière.

Tenez compte des éléments suivants avant d'implémenter une contrainte PRIMARY KEY:

- Une table ne peut posséder qu'une seule contrainte PRIMARY KEY.
- Les colonnes incluses dans la contrainte PRIMARY KEY ne peuvent accepter aucune valeur NULL.
- Les valeurs dans les colonnes spécifiées en tant que clé primaire doivent être uniques. Si une contrainte PRIMARY KEY contient plusieurs colonnes, vous pouvez afficher les doublons dans une seule colonne mais la combinaison des valeurs de toutes les colonnes inscrites dans la définition de contrainte PRIMARY KEY doit être unique.

Une contrainte PRIMARY KEY permet de créer un index unique avec des colonnes spécifiées en tant que clé d'index. Par conséquent, les colonnes choisies pour la clé primaire doivent suivre les règles de création d'index uniques. Vous pouvez spécifier un index cluster ou non-cluster. (L'index cluster constitue la valeur par défaut si aucun index n'existe déjà.) Vous ne pouvez pas supprimer l'index prenant en charge la contrainte PRIMARY KEY. L'index est supprimé uniquement lorsque vous supprimez la contrainte PRIMARY KEY. Cet index permet également d'accéder rapidement aux données lorsque la clé primaire est utilisée dans des requêtes.

**Quand utiliser des contraintes PRIMARY KEY ?**

Envisagez l'utilisation d'une contrainte **PRIMARY KEY** dans les cas suivants:

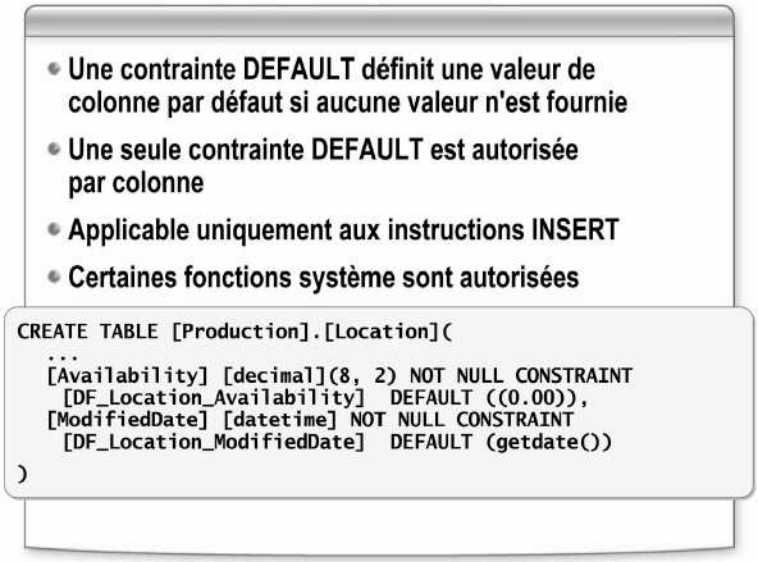
- Une ou plusieurs colonnes dans une table doivent identifier de manière unique chaque ligne (entité) dans la table.
- Une des colonnes de la table est une colonne d'identité.

## 2.1. Création de contraintes PRIMARY KEY

Vous pouvez créer des contraintes PRIMARY KEY à l'aide de la clause CONSTRAINT au niveau de la table des instructions CREATE TABLE et ALTER TABLE. Le code suivant montre l'utilisation de l'instruction Transact-SQL en vue de créer la table **HumanResources.Department** dans la base de données **AdventureWorks**.
La clause CONSTRAINT définit une contrainte PRIMARY KEY cluster appelée **PK_Department_DepartmentID** dans la colonne **DepartmentID**.

```
CREATE TABLE [HumanResources].[Department] (
     [DepartmentID] [smallint] IDENTITY(1,1) NOT NULL, [Name] [dbo].[Name]
     NOT NULL,
    [GroupName] [dbo].[Name] NOT NULL,
    [ModifiedDate] [datetime] NOT NULL,
CONSTRAINT [PK_Department_DepartmentID] PRIMARY KEY CLUSTERED ([DepartmentID] ASC
    )WITH (IGNORE_DUP_KEY = OFF)
ON [PRIMARY] )
```

# 3. Contraintes DEFAULT




**Quand utiliser des contraintes DEFAULT ?**

Envisagez l'utilisation d'une contrainte **DEFAULT** dans les cas suivants:

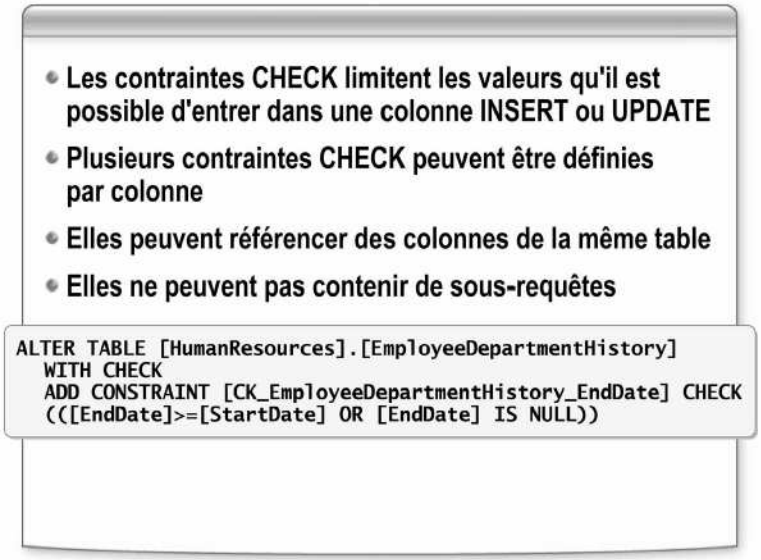
- Les données stockées dans une colonne disposent d'une valeur par défaut évidente.
- La colonne n'accepte pas de valeurs NULL.
- La colonne n'applique pas de valeurs uniques.

**Création de contraintes DEFAULT ?**

Vous pouvez créer des contraintes DEFAULT à l'aide de la clause CONSTRAINT au niveau de la colonne des instructions CREATE TABLE et ALTER TABLE. Le code suivant montre l'utilisation de l'instruction Transact-SQL en vue de créer la table **Production.Location** dans la base de données **AdventureWorks**. L'exemple crée des contraintes DEFAULT dans les colonnes **CostRate**, **Availability** et **ModifiedDate.**

```
CREATE TABLE [Production] . [Location] (
    [LocationID] [smallint] IDENTITY(1,1) NOT NULL,
    [Name] [dbo].[Name] NOT NULL,
    [CostRate] [smallmoney] NOT NULL CONSTRAINT
    [DF_Location_CostRate] DEFAULT ((0.00)),
    [Availability] [decimal](8, 2) NOT NULL
    CONSTRAINT [DF_Location_Availability] DEFAULT
    ((0.00)),
   [ModifiedDate] [datetime] NOT NULL CONSTRAINT
[DF_Location_ModifiedDate] DEFAULT (getdate())
)
```

# 4. Contraintes CHECK




Une contrainte CHECK (contrainte de validation) limite les valeurs de données que les utilisateurs peuvent entrer dans une colonne particulière lors des instructions INSERT et UPDATE. Vous pouvez appliquer des contraintes de validation au niveau de la colonne ou de la table. Les contraintes CHECK au niveau de la colonne limitent les valeurs qu'il est possible de stocker dans une colonne. Les contraintes CHECK au niveau de la table permettent de référencer plusieurs colonnes au sein de la même table pour autoriser un référencement entre les colonnes et leur comparaison.

Tenez compte des éléments suivants avant d'implémenter une contrainte CHECK:

- Une contrainte CHECK vérifie les données chaque fois que vous exécutez une instruction INSERT ou UPDATE.
- Une contrainte CHECK peut être une expression logique (booléenne) quelconque retournant la valeur TRUE ou FALSE.
- Une contrainte CHECK ne peut pas contenir de sous-requêtes.
- Une seule colonne peut comprendre plusieurs contraintes CHECK.
- Vous ne pouvez pas insérer une contrainte CHECK dans des colonnes avec les types de données **rowversion**, **text**, **ntext** ou **image**.

L'instruction CHECKCONSTRAINTS du vérificateur de cohérence de la base de données (DBCC) retourne toutes les lignes contenant les données qui enfreignent une contrainte CHECK.

**Quand utiliser des contraintes CHECK ?**
Envisagez l'utilisation d'une contrainte **CHECK** dans les cas suivants:

- La logique métier veut que les données stockées dans une colonne appartiennent à un ensemble ou une plage spécifique de valeurs.
- Les données stockées dans une colonne fixent des limites naturelles pour les valeurs qu'elles peuvent contenir.

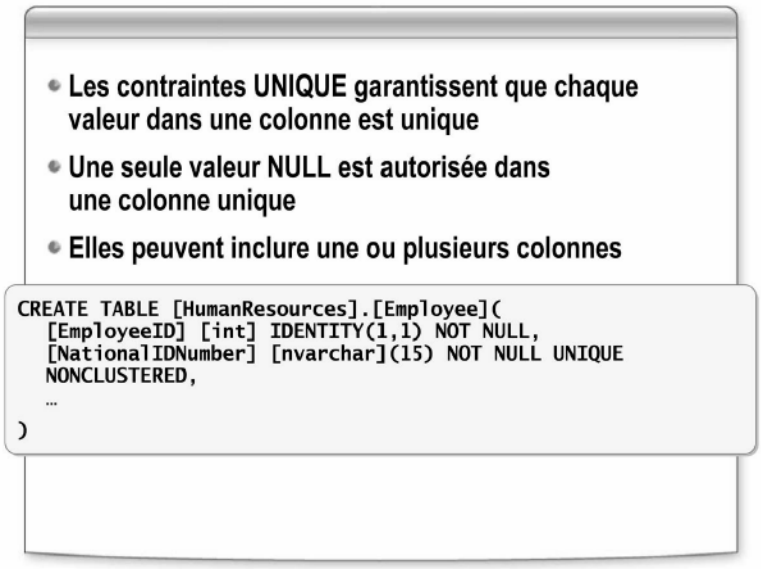
Les relations existant entre les colonnes d'une table restreignent les valeurs que peut contenir une colonne.

**Création de contraintes CHECK ?**
Vous pouvez créer des contraintes CHECK à l'aide de la clause CONSTRAINT au niveau de la colonne ou de la table des instructions CREATE TABLE et ALTER TABLE. Le code suivant montre l'utilisation de l'instruction Transact-SQL en vue d'ajouter une contrainte CHECK à la table **HumanResources.EmployeeDepartmentHistory** de la base de données **AdventureWorks**. L'exemple ci-

dessous montre que la valeur de la colonne **EndDate** est égale ou supérieure à la valeur de la colonne **StartDate** ou que la valeur **EndDate** est NULL.

```
ALTER TABLE [HumanResources].[EmployeeDepartmentHistory] WITH CHECK
ADD CONSTRAINT [CK_Empl oyeeDepartmentHi story_EndDate]
CHECK (([EndDate]>=[StartDate] OR [EndDate] IS NULL))
```

# 5. Contraintes UNIQUE




Une contrainte UNIQUE spécifie que deux lignes dans une colonne ne peuvent pas afficher la même valeur. Une contrainte UNIQUE est utile lorsque vous disposez déjà d'une clé primaire (par exemple, un numéro d'employé) mais souhaitez vous assurer que d'autres identificateurs (par exemple, le numéro fiscal d'un employé) sont également uniques.

Tenez compte des éléments suivants avant d'implémenter une contrainte UNIQUE:

- Une seule valeur NULL peut apparaître dans une colonne avec une contrainte UNIQUE.
- Une table peut posséder plusieurs contraintes UNIQUE mais ne peut disposer que d'une seule clé primaire.
- Vous pouvez appliquer une contrainte UNIQUE en créant un index unique sur la ou les colonnes spécifiées. Cet index ne peut pas permettre à la table de dépasser la limite de 249 sur les index non-cluster.
- Le moteur de base de données renvoie une erreur si vous créez une contrainte UNIQUE dans une colonne contenant des données où apparaissent des valeurs en double.

**Quand utiliser des contraintes UNIQUE ?**

Envisagez l'utilisation d'une contrainte **UNIQUE** dans les cas suivants:

- La table contient des colonnes qui ne font pas partie de la clé primaire mais qui individuellement ou en tant qu'unité, doivent contenir des valeurs uniques.
- La logique métier veut que les données stockées dans une colonne soient uniques.

Les données stockées dans une colonne présentent par nature un caractère unique pour les valeurs qu'elles peuvent contenir (par exemple, des numéros fiscaux ou de passeport).

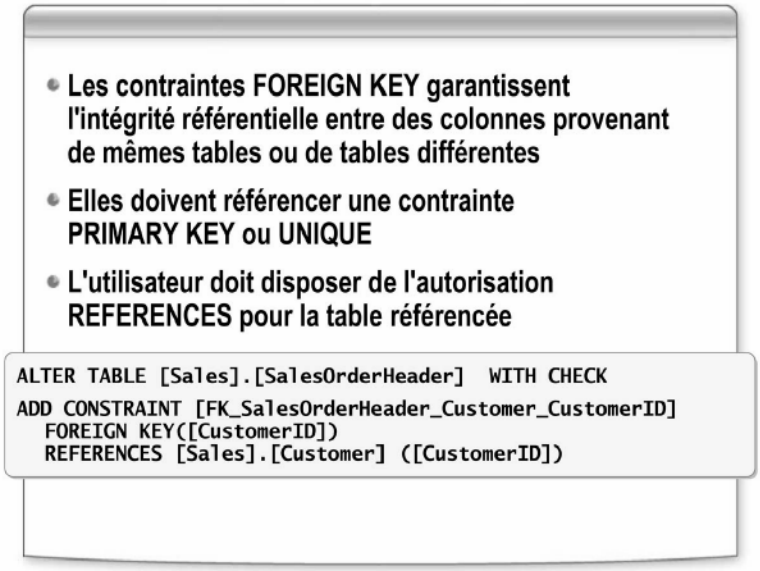
**Création de contraintes UNIQUE ?**

Vous pouvez créer des contraintes UNIQUE à l'aide de la clause UNIQUE au niveau de la colonne des instructions CREATE TABLE et ALTER TABLE. Le code suivant montre l'utilisation de l'instruction Transact-SQL en vue de créer la table **HumanResources.Employee** de la base de données **AdventureWorks**.

L'exemple définit une contrainte UNIQUE dans la colonne **NationalIDNumber** ; il garantit également que la colonne ne contient aucune valeur NULL et que l'index créé pour la prise en charge de la contrainte UNIQUE est un index non-cluster.

```
CREATE TABLE [HumanResources].[Employee] (
[EmployeeID] [int] IDENTITY(1,1) NOT NULL,
[NationalIDNumber] [nvarchar](15) NOT NULL UNIQUE NONCLUSTERED, [ContactID] [int]
NOT NULL,
                    …
)
```

# 6. Contraintes FOREIGN KEY




Le terme « clé étrangère » désigne une colonne ou une combinaison de colonnes utilisée pour établir et utiliser une liaison entre les données de deux tables. Une contrainte FOREIGN KEY permet d'appliquer cette intégrité référentielle. La contrainte FOREIGN KEY définit une référence à une colonne dotée d'une contrainte PRIMARY KEY ou UNIQUE dans la même table ou une table différente. Les valeurs dans la colonne de clé étrangère doivent apparaître dans la colonne de clé primaire.

Si des références aux valeurs de clé primaire existent, il vous est néanmoins impossible de les modifier ou de les supprimer.

Tenez compte des éléments suivants avant d'implémenter une contrainte FOREIGN KEY:

- Une contrainte FOREIGN KEY offre une intégrité référentielle d'une seule ou plusieurs colonnes. Le nombre de colonnes et de types de données spécifiés dans l'instruction FOREIGN KEY doit correspondre au nombre de colonnes et de types de données dans la clause REFERENCES.
- Contrairement aux contraintes PRIMARY KEY ou UNIQUE, les contraintes FOREIGN KEY ne créent pas automatiquement des index.
- Pour qu'un autre utilisateur puisse créer une contrainte FOREIGN KEY dans une table dont vous êtes propriétaire, vous devez lui accorder l'autorisation REFERENCES pour cette table. Ceci garantit qu'aucun autre utilisateur ne peut limiter les opérations dans une table qui vous appartient en créant une contrainte FOREIGN KEY qui référence votre table.
- Une contrainte FOREIGN KEY qui utilise uniquement la clause REFERENCES sans la clause FOREIGN KEY fait référence à une colonne dans la même table.

**Quand utiliser des contraintes UNIQUE ?**
Envisagez l'utilisation d'une contrainte **FOREIGN KEY** dans les cas suivants:

- Les données d'une ou plusieurs colonnes peuvent contenir uniquement des valeurs figurant dans certaines colonnes au sein de la même table ou d'une autre table.

Vous ne devez pas supprimer les lignes d'une table tant que les lignes d'une autre table dépendent d'elles.

**Création de contraintes UNIQUE ?**

Vous pouvez créer des contraintes FOREIGN KEY à l'aide de la clause CON STRAINT au niveau de la colonne ou de la table des instructions CREATE TABLE et ALTER TABLE. Le code suivant montre l'utilisation de l'instruction Transact-SQL en vue d'ajouter une contrainte FOREIGN KEY à la table **Sales.SalesOrderHeader** de la base de données **AdventureWorks**. L'exemple crée une contrainte FOREIGN KEY appelée **FK_SalesOrderHeader_Customer_CustomerID** qui établit un lien référentiel entre la colonne **CustomerID** de la table **Sales.SalesOrderHeader** et la colonne **CustomerID** de la table **Sales.Customer**.

```
ALTER TABLE [Sales].[SalesOrderHeader] WITH CHECK ADD CONSTRAINT
[FK_SalesOrderHeader_Customer_CustomerID]
FOREIGN KEY([CustomerID])
REFERENCES [Sales].[Customer] ([CustomerID])
```

# 7. Intégrité référentielle en cascade

**Contrôlée par la clause CASCADE de la contrainte FOREIGN KEY**

| Option CASCADE | Comportement UPDATE | Comportement DELETE |
|---|---|---|
| NO ACTION (valeur par défaut) | erreur générée, opération restaurée | |
| CASCADE | Clés étrangères mises à jour dans les tables de référence | Lignes supprimées dans les tables de référence |
| SET NULL | Clés étrangères définies dans les tables de référence sur des valeurs NULL | |
| SET DEFAULT | Clés étrangères définies dans les tables de référence sur des valeurs par défaut | |

La contrainte FOREIGN KEY comprend une option CASCADE qui vous permet de modifier une valeur de colonne définissant une contrainte UNIQUE ou PRIMARY KEY et d'étendre cette modification à toutes les valeurs de clé étrangère qui la référencent. On emploie le terme d'intégrité référentielle en cascade pour désigner cette action.
Les clauses REFERENCES des instructions CREATE TABLE et ALTER TABLE prennent en charge les clauses ON DELETE et ON UPDATE: Ces clauses vous permettent de spécifier le comportement de l'intégrité référentielle en cascade. Les valeurs possibles incluent notamment les valeurs NO ACTION, CASCADE, SET NULL et SET DEFAULT NO ACTION est la valeur par défaut.

## 7.1. Mise à jour en cascade

La clause ON UPDATE de la contrainte FOREIGN KEY définit le comportement
Lorsqu'une instruction UPDATE tente de modifier une valeur de clé primaire référencée par des valeurs de clé étrangère provenant d'autres tables. Le tableau suivant résume les valeurs possibles de la clause ON UPDATE et leurs effets.

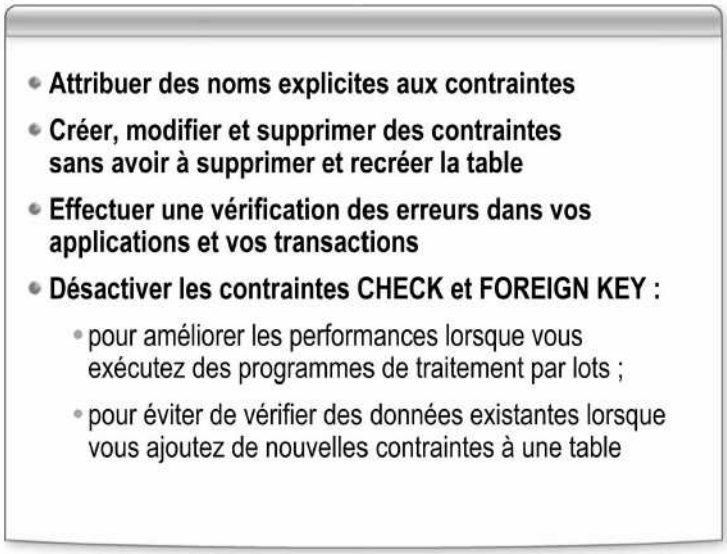
| Option | Comportement |
|---|---|
| NO ACTION | Spécifie qu'en cas de tentative de mise à jour d'une valeur de clé dans une ligne dont la clé est référencée par des clés étrangères dans les lignes d'autres tables, une erreur est générée et l'instruction UPDATE est restaurée. |
| CASCADE | Spécifie qu'en cas de tentative de mise à jour, dans une ligne, d'une valeur de clé référencée par des clés étrangères dans les lignes d'autres tables, toutes les valeurs qui composent la clé étrangère sont également mises à jour et remplacées par la nouvelle valeur spécifiée pour la clé. |
| SET NULL | Spécifie qu'en cas de tentative de mise à jour d'une ligne à l'aide d'une clé référencée par des clés étrangères dans les lignes d'autres tables, toutes les valeurs qui composent la clé étrangère dans les lignes référencées prennent la valeur NULL. Vous pouvez exécuter cette contrainte uniquement si toutes les colonnes de clé étrangère de la table cible acceptent des valeurs NULL. |
| SET DEFAULT | Spécifie qu'en cas de tentative de mise à jour d'une ligne à l'aide d'une clé référencée par des clés étrangères dans les lignes d'autres tables, toutes les valeurs qui composent la clé étrangère dans les lignes référencées prennent leur valeur par défaut. Cette contrainte ne peut être exécutée que si toutes les colonnes de clé étrangère de la table cible adoptent une définition par défaut. Si une colonne est affectée de la valeur NULL et qu'aucune valeur par défaut explicite n'est définie, NULL constitue alors la valeur par défaut implicite de la colonne. La validité de la contrainte FOREIGN KEY ne peut être conservée que s'il existe dans la table primaire des valeurs correspondant aux valeurs non NULL définies conformément à l'action ON UPDATE SET DEFAULT. |

## 7.2.Suppression en cascade

La clause ON DELETE de la contrainte FOREIGN KEY définit le comportement lorsqu'une instruction DELETE tente de supprimer une valeur de clé primaire référencée par des valeurs de clé étrangère provenant d'autres tables. Le tableau suivant résume les valeurs possibles de la clause ON DELETE et leurs effets.

| Option | Comportement |
|---|---|
| NO ACTION | Spécifie qu'en cas de tentative de suppression d'une ligne à l'aide d'une clé référencée par des clés étrangères dans les lignes d'autres tables, une erreur est générée et l'instruction DELETE est restaurée. |
| CASCADE | Spécifie qu'en cas de tentative de suppression d'une ligne à l'aide d'une clé référencée par des clés étrangères dans les lignes d'autres tables, toutes les lignes contenant ces clés étrangères sont également supprimées. |
| SET NULL | Spécifie qu'en cas de tentative de suppression d'une ligne à l'aide d'une clé référencée par des clés étrangères dans les lignes d'autres tables, toutes les valeurs qui composent la clé étrangère dans les lignes référencées prennent la valeur NULL. Vous pouvez exécuter cette contrainte uniquement si toutes les colonnes de clé étrangère de la table cible acceptent des valeurs NULL. |
| SET DEFAULT | Spécifie qu'en cas de tentative de suppression d'une ligne à l'aide d'une clé référencée par des clés étrangères dans les lignes d'autres tables, toutes les valeurs qui composent la clé étrangère dans les lignes référencées prennent leur valeur par défaut. Cette contrainte ne peut être exécutée que si toutes les colonnes de clé étrangère de la table cible adoptent une définition par défaut. Si une colonne est affectée de la valeur NULL et qu'aucune valeur par défaut explicite n'est définie, NULL constitue alors la valeur par défaut implicite de la colonne. La validité de la contrainte FOREIGN KEY ne peut être conservée que s'il existe dans la table primaire des valeurs correspondant aux valeurs non NULL définies conformément à l'action ON DELETE SET DEFAULT. |

## 8. Eléments à prendre en compte pour la vérification des contraintes




Vous devez spécifier des noms explicites lorsque vous créez des contraintes ; si vous ne donnez aucun nom à vos contraintes, SQL Server génère des noms système compliqués. Les noms doivent être uniques au propriétaire de l'objet de base de données et suivre les règles définies pour les identificateurs SQL Server.

Tenez compte des faits suivants lorsque vous implémentez ou modifiez des contraintes:

- Vous pouvez créer, modifier et supprimer des contraintes sans avoir à supprimer et recréer une table.

- Vous devez générer une logique de vérification des erreurs dans vos applications et vos transactions afin d'évaluer si une contrainte a été violée.
- Vous devez préciser si vous ne souhaitez pas que SQL Server applique les contraintes FOREIGN KEY et CHECK aux données existantes lorsque vous ajoutez de nouvelles contraintes à une table existante.

## 8.1. Quand désactiver les contraintes

Vous pouvez désactiver uniquement les contraintes CHECK et FOREIGN KEY. Les autres contraintes doivent être supprimées, puis rajoutées. La désactivation des contraintes peut être envisagée dans les situations suivantes:

- Vous devez exécuter un programme de traitement par lots important ou importer des données et souhaitez optimiser les performances. Avant de réactiver les contraintes, vous devez être certain que les données sont conformes aux contraintes appropriées ou exécuter les requêtes dans le cadre du programme de traitement par lots pour garantir que les données sont exactes.
- Vous définissez une contrainte dans une table contenant déjà des données. Chaque ligne de données est alors vérifiée uniquement par la contrainte la prochaine fois que vous la modifiez.

## 8.2. Comment désactiver les contraintes

Pour désactiver la vérification des contraintes lorsque vous ajoutez une contrainte CHECK ou FOREIGN KEY à une table avec des données existantes, incluez l'option WITH NOCHECK dans l'instruction ALTER TABLE. Les données existantes sont vérifiées uniquement lors des prochaines mises à jour de la colonne de contrainte.

L'exemple suivant crée une contrainte FOREIGN KEY dans la table **Sales.SalesOrderHeader** et utilise l'option WITH NOCHECK pour désactiver la vérification des données existantes.

```
ALTER TABLE [Sales].[SalesOrderHeader] WITH NOCHECK ADD CONSTRAINT
[FK_SalesOrderHeader_Customer_CustomerID] FOREIGN KEY([CustomerID]) REFERENCES [Sales] . [Customer]
([CustomerID])
```

Vous pouvez désactiver la vérification des contraintes CHECK et FOREIGN KEY existantes afin que toutes les données que vous modifiez ou ajoutez à la table ne soient pas vérifiées par rapport à la contrainte. Pour désactiver une contrainte CHECK et FOREIGN KEY, utilisez l'instruction ALTER TABLE et spécifiez l'option NOCHECK. Pour activer une contrainte désactivée, exécutez une autre instruction ALTER TABLE, cette fois avec l'option CHECK.

L'exemple ci-dessous désactive la contrainte **FK_SalesOrderHeader_Customer_CustomerID**.

```
ALTER TABLE [Sales]. [SalesOrderHeader]
NOCHECK
CONSTRAINT [FK_SalesOrderHeader_Customer_CustomerID]
```

L'exemple ci-dessous réactive la contrainte **FK_SalesOrderHeader_Customer_CustomerID**.

```
ALTER TABLE [Sales]. [SalesOrderHeader]
CHECK
CONSTRAINT [FK_SalesOrderHeader_Customer_CustomerID]
```

# 9. Application pratique : Création de contraintes

## 9.1. Suppression des contraintes existantes

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes** et **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.

2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Dans le menu **Fichier**, pointez sur **Ouvrir**, puis cliquez sur **Fichier**.
4. Accédez au dossier **D:\Practices**, puis ouvrez le fichier **DropConstraints.sql**.
5. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
6. Fermez la requête **DropConstraints.sql**.

## 9.2.Créer une contrainte CHECK à l'aide de Transact-SQL

Vous devez exécuter les étapes suivantes pour créer une contrainte CHECK à l'aide de Transact-SQL:

1. Dans SQL Server Management Studio, si l'Explorateur d'objets n'apparaît pas, cliquez sur **Explorateur d'objets** dans le menu **Affichage**.
2. Dans l'Explorateur d'objets, développez **Bases de données**, **AdventureWorks**, **Tables**, **HumanResources.Employee** et **Contraintes** pour afficher les contraintes de la table **HumanResources.Employee**.
3. Cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
4. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks
GO
ALTER TABLE HumanResources.Employee WITH CHECK ADD CONSTRAINT CK_Employee_Gender
     CHECK ((upper([Gender])=' F' OR upper([Gender])='M'))
```

5. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
6. Une fois l'exécution réussie, cliquez avec le bouton droit sur le dossier **Contraintes** dans l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que la contrainte **CK_Employee_Gender** a été créée.
7. Cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
8. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks
GO
UPDATE HumanResources.Employee
SET Gender = 'Q'
WHERE EmployeeID = 1
```

9. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
10. Vérifiez si une erreur a été retournée puisque la mise à jour était en conflit avec la contrainte CHECK.
11. Laissez SQL Server Management Studio ouvert. Vous allez l'utiliser au cours de la procédure suivante.

## 9.3.Créer une contrainte CHECK à l'aide de Transact-SQL

Vous devez exécuter les étapes suivantes pour créer une contrainte DEFAULT à l'aide de Transact-SQL:

1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** de la barre d'outils.
2. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks
GO
ALTER TABLE HumanResources.Employee ADD CONSTRAINT DF_Employee_ModifiedDate
     DEFAULT (getdate()) FOR ModifiedDate
```

3. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
4. Une fois l'exécution réussie, cliquez avec le bouton droit sur le dossier **Contraintes** dans l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que la contrainte **DF_Employee_ModifiedDate** a été créée.
5. Cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
6. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

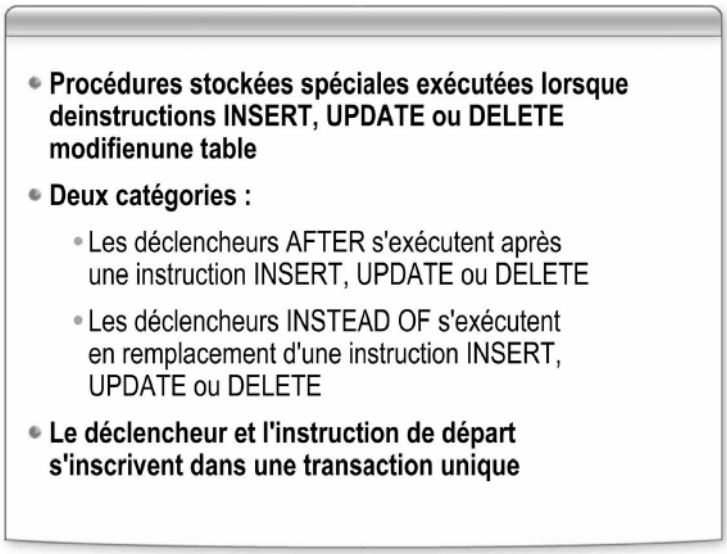
```
USE AdventureWorks
GO
UPDATE HumanResources.Employee
SET ModifiedDate = DEFAULT
WHERE EmployeeID = 1

SELECT ModifiedDate
FROM HumanResources.Employee
WHERE EmployeeID = 1
```

7. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
8. Vérifiez que la valeur par défaut **ModifiedDate** correspond à la valeur de date et heure actuelle.
9. Fermez SQL Server Management Studio. Cliquez sur **Non** si le programme vous demande d'enregistrer les fichiers.

# Leçon 3 : Implémentation de déclencheurs

## 1. Que sont les déclencheurs ?




Un déclencheur est une forme spéciale de procédure stockée exécutée lorsqu'une instruction INSERT, UPDATE ou DELETE modifie les données d'une table précise.
Un déclencheur peut interroger d'autres tables et peut inclure des instructions TransactSQL complexes. Les déclencheurs sont souvent créés pour assurer l'intégrité référentielle ou la cohérence de données reliées de manière logique dans différentes tables. Du fait du caractère incontournable des déclencheurs pour les utilisateurs et de l'accès aux fonctionnalités de Transact-SQL dont vous bénéficiez, vous pouvez utiliser les déclencheurs pour appliquer une logique métier complexe difficile ou impossible à appliquer à l'aide d'autres mécanismes d'intégrité des données.

Tenez compte des points suivants à propos des déclencheurs:

- Le déclencheur et l'instruction qui le déclenche sont traités comme une transaction unique

qui peut être annulée (par une opération de restauration) à partir du déclencheur. Si une erreur grave est détectée (par exemple, un espace disque insuffisant), la transaction tout entière est automatiquement annulée.

- Les déclencheurs peuvent effectuer des modifications en cascade dans des tables associées de la base de données, mais ces modifications peuvent être réalisées de manière plus efficace à l'aide de contraintes d'intégrité référentielle en cascade.

- Ils peuvent vous protéger contre des opérations d'insertion, de mise à jour et de suppression malveillantes ou incorrectes et garantir l'application d'autres restrictions plus complexes que celles définies à l'aide des contraintes CHECK.
Contrairement aux contraintes CHECK, les déclencheurs peuvent référencer des colonnes dans d'autres tables. Par exemple, un déclencheur peut utiliser une instruction SELECT à partir d'une autre table en vue d'une comparaison avec des données insérées ou mises à jour et d'actions supplémentaires (par exemple, la modification des données ou l'affichage d'un message d'erreur défini par l'utilisateur).

- Vous pouvez recourir aux déclencheurs pour évaluer l'état d'une table avant et après avoir modifié des données et entreprendre des actions en fonction des différences constatées.

- Plusieurs déclencheurs du même type (INSERT, UPDATE ou DELETE) dans une table permettent d'entreprendre plusieurs actions différentes en réponse à la même instruction de modification.

## 1.1. Création de déclencheurs

Vous pouvez créer des déclencheurs à l'aide de l'instruction Transact-SQL CREATE TRIGGER.
L'instruction CREATE TRIGGER présente la syntaxe suivante :

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ...n ] | EXTERNAL NAME <method specifier [ ; ] > }
```

## 1.2. Types de déclencheurs

Il existe deux types de déclencheurs DML:

***Déclencheurs AFTER***. Les déclencheurs AFTER sont exécutés après l'action associée à une instruction INSERT, UPDATE ou DELETE. Le déclencheur AFTER est défini de la même manière que le déclencheur FOR, ce qui constitue la seule option disponible dans les versions antérieures de Microsoft SQL Server. Vous pouvez définir des déclencheurs AFTER uniquement dans les tables.

***Déclencheurs INSTEAD OF***. Les déclencheurs INSTEAD OF sont exécutés à la place de l'action de déclenchement habituelle. Ils peuvent également être définis dans des vues avec une ou plusieurs tables de base afin d'étendre les types de mises à jour qu'une vue peut prendre en charge.

## 1.3. Déclencheurs ou contraintes

Le principal atout des déclencheurs est qu'ils peuvent présenter une logique de traitement complexe fondée sur le code Transact-SQL. Ils permettent donc la prise en charge d'un sur-ensemble des fonctionnalités des contraintes. Toutefois, tant que les contraintes répondent aux exigences de la solution en terme de configuration requise, l'intégrité des données doit rester appliquée au niveau le plus bas possible à l'aide de contraintes.

Les déclencheurs sont particulièrement utiles lorsque les fonctionnalités prises en charge par les contraintes ne peuvent pas répondre aux besoins fonctionnels de l'application.

Exemple:

• Les contraintes FOREIGN KEY peuvent valider une valeur de colonne uniquement si celle-ci correspond exactement à la valeur d'une autre colonne, sauf si la clause REFERENCES définit une action d'intégrité référentielle en cascade.

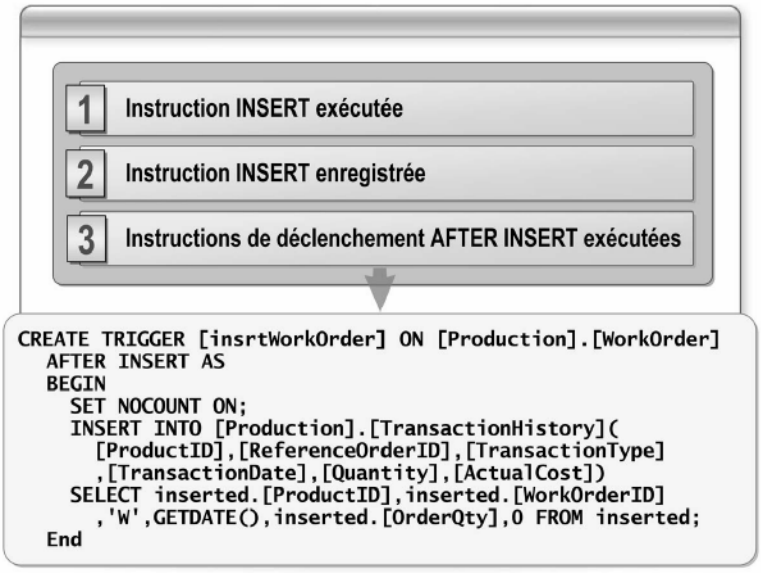
• Les contraintes peuvent fournir des informations sur les erreurs uniquement par le biais de messages d'erreur système standard. Si votre application exige un mode de traitement plus complexe des erreurs, vous devez utiliser un déclencheur.

Les déclencheurs peuvent effectuer des modifications en cascade dans des tables associées de la base de données, mais ces modifications peuvent être réalisées de manière plus efficace à l'aide de contraintes d'intégrité référentielle en cascade.

• Les déclencheurs peuvent servir à interdire ou à annuler par restauration des modifications qui violent l'intégrité référentielle, annulant ainsi la tentative de modification des données. Un déclencheur de ce type peut entrer en action si vous modifiez une clé étrangère et si la nouvelle valeur ne correspond pas à sa clé primaire. Toutefois, les contraintes FOREIGN KEY sont généralement utilisées dans ce but.

• Si des contraintes existent dans la table du déclencheur, elles sont vérifiées après l'exécution du déclencheur INSTEAD OF mais avant celle du déclencheur AFTER. En cas de violation des contraintes, les actions du déclencheur INSTEAD OF sont restaurées et le déclencheur AFTER n'est pas exécuté.

## 2. Mode de fonctionnement d'un déclencheur INSERT




Un déclencheur INSERT s'exécute lorsqu'une instruction INSERT insère des données dans une table ou une vue dans laquelle le déclencheur est configuré.

Lors de l'application d'un déclencheur INSERT, des nouvelles lignes sont ajoutées à la fois à la table du déclencheur et à la table **insérée**. La table **insérée** est une table logique qui contient une copie des lignes qui ont été insérées. Cette table **insérée** sert également à enregistrer les activités d'insertion de l'instruction INSERT. Elle vous permet de référencer les données enregistrées de l'instruction INSERT de départ. Le déclencheur peut examiner la table insérée afin de déterminer si, ou comment, les actions du déclencheur doivent être exécutées. Les lignes de la table **insérée** sont toujours des doublons d'une ou plusieurs lignes créées dans la table du déclencheur.

Toutes les activités de modification des données (instructions INSERT, UPDATE et DELETE) sont enregistrées mais les informations du journal des transactions sont illisibles. En revanche, la table **insérée** vous permet de référencer les modifications enregistrées issues de l'instruction INSERT. Vous pouvez alors comparer les modifications avec les données insérées pour les vérifier ou entreprendre une autre action. De même, vous pouvez référencer des données insérées sans avoir à stocker les informations dans des variables.

## 2.1. Implémentation d'un déclencheur INSERT

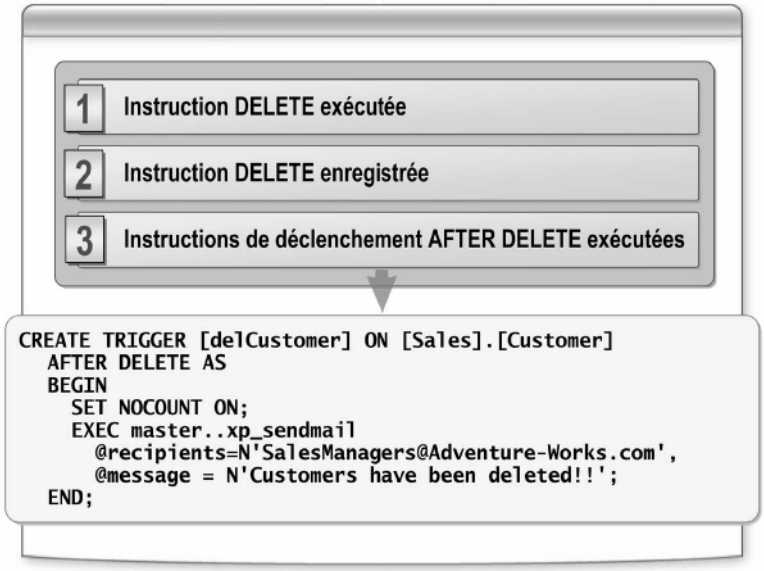
Le code ci-dessous explique la création d'un déclencheur INSERT nommé **insrtWorkOrder** dans la table **Production.WorkOrder** de la base de données **AdventureWorks**. Notez l'utilisation de la table **insérée** en vue d'exploiter les valeurs qui ont provoqué l'exécution du déclencheur.

```
CREATE TRIGGER [insrtWorkOrder] ON [Production].[WorkOrder]
AFTER INSERT AS
BEGIN
SET NOCOUNT ON;

INSERT INTO [Production].[TransactionHistory](
        [ProductID],[ReferenceOrderID],
        [TransactionType],[TransactionDate],[Quantity],[ActualCost])
SELECT inserted.[ProductID],inserted.[WorkOrderID],'W',GETDATE()
,inserted.[OrderQty], 0
FROM inserted;

END;
```

# 3. Mode de fonctionnement d'un déclencheur DELETE




Un déclencheur DELETE est une forme spéciale de procédure stockée exécutée Lors de l'application d'un déclencheur DELETE, les lignes supprimées dans la table concernée sont placées dans une table **supprimée** spéciale. La table **supprimée** est une table logique qui contient une copie des lignes qui ont été supprimées. Elle vous permet de référencer les données enregistrées de l'instruction DELETE de départ.

Tenez compte des points suivants lorsque vous utilisez le déclencheur DELETE:

- Lorsqu'une ligne est ajoutée à la table **supprimée**, elle disparaît dans la table de base de données ; la table **supprimée** et les tables de base de données n'ont donc aucune ligne en commun.
- L'espace mémoire alloué permet de créer la table **supprimée**. La table **supprimée** est conservée en permanence dans le cache.
- Un déclencheur défini pour une action DELETE n'est pas exécuté pour l'instruction TRUNCATE

TABLE puisque cette dernière n'est pas enregistrée.lorsqu'une instruction DELETE supprime les données d'une table ou d'une vue dans laquelle le déclencheur est configuré.

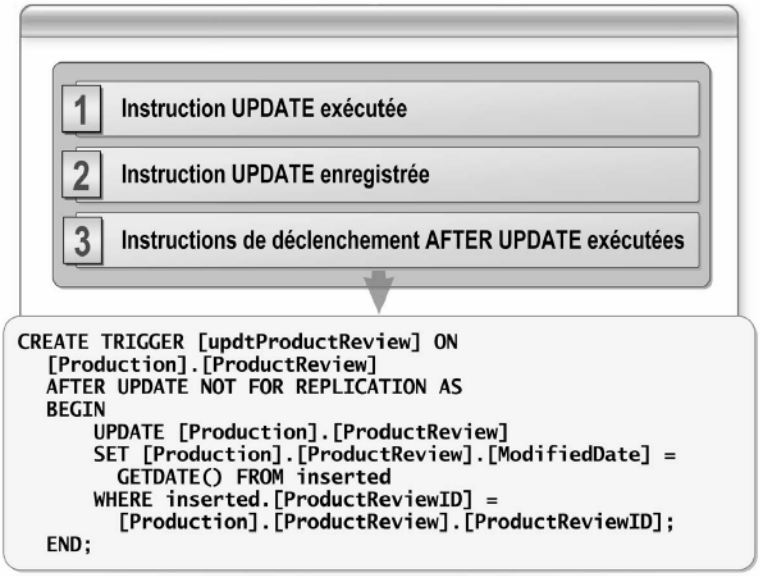
### 3.1. Implémentation d'un déclencheur DELETE

Le code ci-dessous explique la création d'un déclencheur DELETE nommé **delCustomer** dans la table **Sales.Customer** de la base de données **AdventureWorks**.

```
 CREATE TRIGGER [delCustomer] ON [Sales] . [Customer] AFTER DELETE AS
BEGIN
  SET NOCOUNT ON;

  EXEC master. .xp_sendmail
     @recipients='SalesManagers@Adventure-Works.com',
     @message = 'Customers have been deleted!!';
END;
```

## 4. Mode de fonctionnement d'un déclencheur UPDATE




Un déclencheur UPDATE s'exécute lorsqu'une instruction UPDATE modifie les données d'une table ou d'une vue dans laquelle le déclencheur est configuré.

Un déclencheur UPDATE peut se résumer à deux étapes:

1. L'étape DELETE chargée de capturer l'*image de départ* des données.
2. L'étape INSERT chargée de capturer l'image finale des données.

Lorsque vous exécutez une instruction UPDATE dans une table pour laquelle un déclencheur a été défini, les lignes d'origine (image de départ) sont déplacées dans la table **supprimée** et les lignes mises à jour (image finale) sont insérées dans la table **insérée**.

Le déclencheur peut examiner les tables **supprimée** et **insérée**, ainsi que la table **mise à jour**, afin de déterminer si plusieurs lignes ont été mises à jour et comment effectuer ses actions.

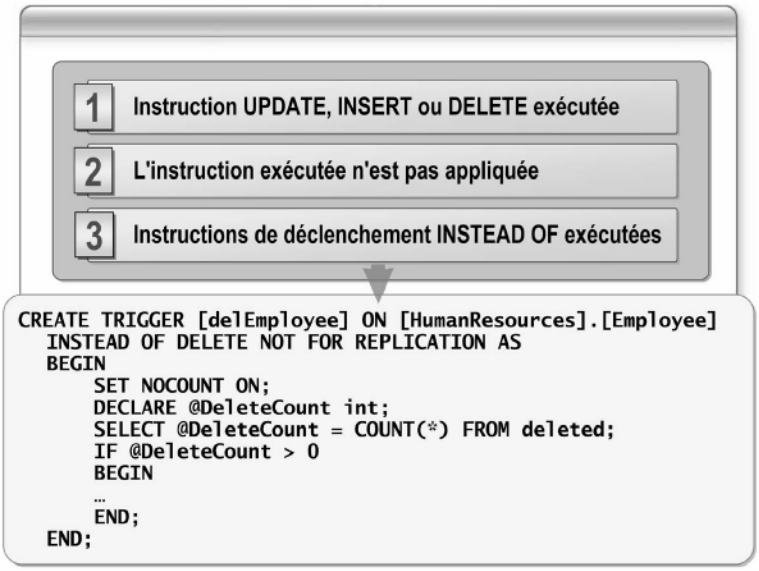
Vous pouvez définir un déclencheur pour surveiller les mises à jour des données dans une colonne spécifique à l'aide de l'instruction IF UPDATE. Cela permet au déclencheur d'isoler facilement les activités d'une colonne spécifique. Lorsqu'il détecte que la colonne spécifique a été mise à jour, il peut entreprendre l'action adéquate, notamment en affichant un message d'erreur qui indique que la colonne ne peut pas être mise à jour ou en procédant au traitement d'une série d'instructions sur la base de la nouvelle valeur de colonne mise à jour.

### 4.1. Implémentation d'un déclencheur UPDATE

Le code ci-dessous explique la création d'un déclencheur UPDATE nommé **updtProductReview** dans la table **Production.ProductReview** de la base de données **AdventureWorks**.

```
CREATE TRIGGER [updtProductReview] ON [Production].[ProductReview] AFTER UPDATE NOT FOR
REPLICATION AS
BEGIN
        SET NOCOUNT ON;
        UPDATE [Production].[ProductReview]
        SET [Production].[ProductReview].[ModifiedDate] = GETDATE() FROM inserted
        WHERE inserted.[ProductReviewID] = [Production].[ProductReview].[ProductReviewID];
```

## 5. Mode de fonctionnement d'un déclencheur INSTEAD OF




Les déclencheurs INSTEAD OF sont exécutés à la place de l'action de déclenchement habituelle. Ils peuvent également être définis dans des vues avec une ou plusieurs tables de base afin d'étendre les types de mises à jour qu'une vue peut prendre en charge.

Ce déclencheur s'exécute à la place de l'action de déclenchement d'origine. Les déclencheurs INSTEAD OF augmentent la diversité des types de mises à jour que vous pouvez effectuer dans une vue. Chaque table ou vue est limitée à un déclencheur INSTEAD OF par action de déclenchement (INSERT, UPDATE ou DELETE).
Vous pouvez spécifier un déclencheur INSTEAD OF à la fois dans des tables et dans des vues. Vous ne pouvez pas créer un déclencheur INSTEAD OF dans des vues pour lesquelles l'option WITH CHECK OPTION est définie.

La liste suivante présente les avantages principaux des déclencheurs INSTEAD OF:

- Ils permettent la prise en charge d'insertions, de mises à jour et de suppressions référençant des données de tables dans des vues dotées de plusieurs tables de base.
- Ils vous permettent de coder une logique capable de rejeter certaines parties d'un programme de traitement par lots et d'en mener d'autres à terme.

Ils vous permettent de spécifier une autre action de base de données dans des situations respectant des conditions précises.

### 5.1. Implémentation d'un déclencheur INSTEAD OF

Le code ci-dessous explique la création d'un déclencheur INSTEAD OF nommé **delEmployee** dans la table **HumanResources.Employee** de la base de données **AdventureWorks**.

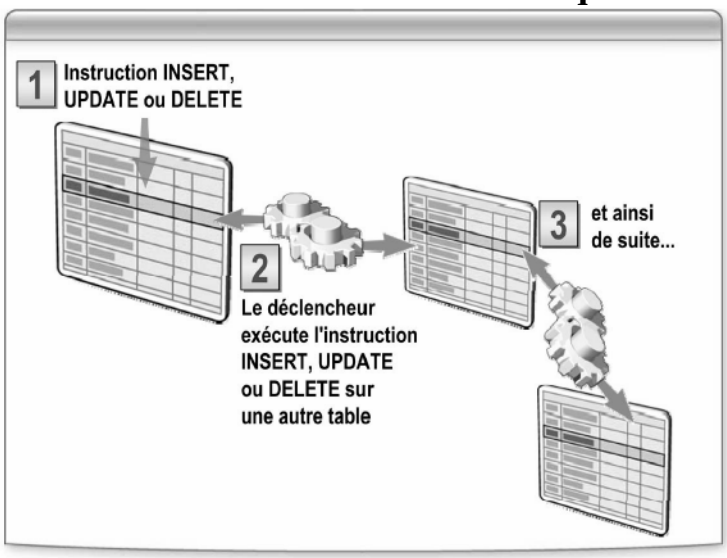
```
CREATE TRIGGER [delEmployee] ON [HumanResources].[Employee]
INSTEAD OF DELETE NOT FOR REPLICATION AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @DeleteCount int;

    SELECT @DeleteCount = COUNT(*) FROM deleted;
    IF @DeleteCount > 0
    BEGIN
        RAISERROR
            (N'Employees cannot be deleted. They can only be marked as not
current.', -- Message
            10, -- Severity.
            1); -- State.

        -- Roll back any active or uncommittable transactions
        IF @@TRANCOUNT > 0
        BEGIN
            ROLLBACK TRANSACTION;
        END
     END;
END;
```

# 6. Mode de fonctionnement de déclencheurs imbriqués




Tous les déclencheurs peuvent contenir une instruction UPDATE, INSERT ou DELETE ayant une incidence sur une autre table. Les déclencheurs sont imbriqués lorsqu'un déclencheur exécute une action qui lance un autre déclencheur.

Vous pouvez déterminer si des déclencheurs sont imbriqués à l'aide de l'option de configuration des déclencheurs imbriqués du serveur La fonction d'imbrication est activée lors de l'installation et définie au niveau du serveur mais vous pouvez la désactiver et la réactiver à l'aide de la procédure système stockée **sp_configure**.

Les déclencheurs peuvent contenir jusqu'à 32 niveaux d'imbrication. Lorsqu'un déclencheur dans une chaîne imbriquée provoque une boucle infinie, la capacité d'imbrication est dépassée. Le déclencheur s'arrête alors et restaure la transaction. Vous pouvez recourir aux déclencheurs imbriqués pour la réalisation de certaines fonctions, notamment pour sauvegarder une copie de lignes affectées par un déclencheur précédent.

Tenez compte des points suivants lorsque vous utilisez des déclencheurs imbriqués:

- Par défaut, l'option de configuration d'un déclencheur imbriqué est activée.
- Un déclencheur imbriqué ne se déclenche pas deux fois dans le cadre de la même transaction de déclenchement ; un déclencheur ne s'applique pas de lui-même en réponse à une deuxième mise à

jour de la même table au sein du déclencheur. En revanche, si un déclencheur modifie une table qui entraîne l'application d'un autre déclencheur et si ce deuxième déclencheur modifie la table d'origine, le déclencheur de départ s'appliquera de manière récursive. Pour éviter ce phénomène de récursivité indirecte, désactivez l'option des déclencheurs imbriqués.

- Un déclencheur étant une transaction, tout échec à un niveau quelconque d'un ensemble de déclencheurs imbriqués entraîne l'annulation de la transaction tout entière et toutes les modifications apportées aux données sont restaurées. Par conséquent, il est préférable d'inclure des instructions PRINT lorsque vous testez des déclencheurs afin de déterminer la source de l'échec.

## 6.1.Niveaux d'imbrication

Le niveau d'imbrication augmente chaque fois que le déclencheur imbriqué est appliqué. SQL Server prend en charge un nombre maximal de 32 niveaux d'imbrication mais vous pouvez à votre guise limiter ces niveaux pour éviter de dépasser le niveau d'imbrication maximal. Vous pouvez afficher vos niveaux d'imbrication actuels à l'aide de la fonction @@NESTLEVEL.

## 6.2.Désactivation des déclencheurs imbriqués

L'imbrication est une fonctionnalité puissante qui vous permet de protéger l'intégrité des données dans une base de données tout entière. Toutefois, de temps à autre, vous pouvez souhaiter la désactiver. En désactivant la fonction d'imbrication, vous ne pouvez pas appliquer un déclencheur en cascade (action permettant de lancer un déclencheur qui entraîne un autre déclencheur, et ainsi de suite).

Vous pouvez avoir besoin de désactiver l'imbrication pour les raisons suivantes:

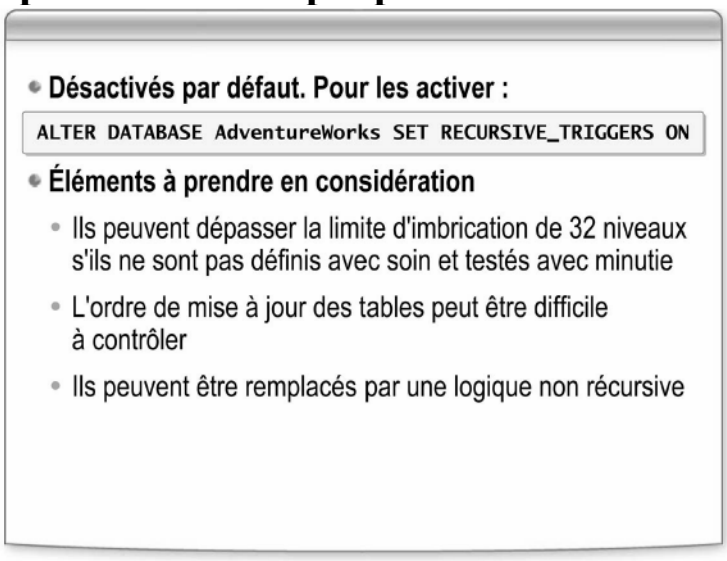
- Les déclencheurs imbriqués exigent une structure complexe et planifiée avec soin. Les modifications en cascade peuvent entraîner des modifications de données involontaires.

- Toute modification de données survenant à un point donné dans une série de déclencheurs imbriqués provoque le déclenchement de cette série. Malgré l'avantage que confère cette situation pour la protection de vos données, elle peut poser un problème si vos tables doivent être mises à jour dans un ordre précis.

Utilisez l'instruction suivante pour désactiver l'imbrication.

```
sp_configure 'nested triggers', 0
```

# 7. Eléments à prendre en compte pour les déclencheurs récursifs




Un déclencheur récursif exécute une action qui provoque la réapplication de ce même déclencheur de manière directe ou indirecte. Un déclencheur peut contenir une instruction UPDATE, INSERT ou DELETE

qui affecte la même table ou une autre table. Avec l'option de **déclencheur récursif** activée, un déclencheur qui modifie les données d'une table est en mesure de s'activer lui-même dans le cadre d'une exécution récursive.

Il existe deux types de récursivité:

***Récursivité directe***. La récursivité directe survient lorsqu'un déclencheur est activé et exécute une action dans la même table qui l'active de nouveau. Par exemple, une application met à jour la table **T1**, ce qui active le déclencheur **Trig1**. **Trig1** met à jour de nouveau la table **T1**, ce qui réactive le déclencheur **Trig1**.

***Récursivité indirecte***. Il y a récursivité indirecte lorsqu'un déclencheur est activé et exécute une action qui active un autre déclencheur (dans la même table ou une autre table) entraînant ainsi la mise à jour de la table d'origine. Le déclencheur d'origine est alors de nouveau activé. Par exemple, une application met à jour la table **T2**, ce qui active le déclencheur **Trig2**. **Trig2** met à jour de nouveau la table **T3**, ce qui réactive le déclencheur **Trig3**. **Trig3**, à son tour, met à jour la table **T2**, ce qui réactive le déclencheur **Trig2**.

## 7.1.Comment contrôler le comportement d'appel récursif

L'option de déclencheur récursif est désactivée par défaut lorsque vous créez une base de données, mais vous pouvez l'activer à l'aide de l'instruction ALTER DATABASE. Un déclencheur ne s'appelle pas lui-même de manière récursive, sauf si l'option de base de données RECURSIVE_TRIGGERS est définie. Utilisez l'instruction suivante pour activer des déclencheurs récursifs.

```
ALTER DATABASE AdventureWorks SET RECURSIVE_TRIGGERS ON
sp_dboption databasename, 'recursive triggers', True
```

Si l'option de déclencheur imbriqué est désactivée, l'option de déclencheur récursif l'est également, quel que soit le paramètre de déclencheur récursif défini dans la base de données.

Si l'option de déclencheur imbriqué est activée, la récursivité indirecte continue de s'appliquer même si l'option récursive est désactivée. L'option de déclencheur récursif protége uniquement contre la récursivité directe. (Voir les définitions de récursivité directe et indirecte plus haut dans cette rubrique.)

Les tables **insérée** et **supprimée** d'un déclencheur donné contiennent des lignes qui correspondent uniquement à la dernière instruction UPDATE, INSERT ou DELETE ayant appelé le déclencheur.

La récursivité des déclencheurs peut aller jusqu'à 32 niveaux. Si la récursivité d'un déclencheur provoque une boucle infinie dès qu'il dépasse la limite de 32 niveaux, le déclencheur s'arrête automatiquement et la transaction est restaurée.

## 7.2.Eléments à prendre en compte pour les déclencheurs récursifs

Les déclencheurs récursifs sont une fonctionnalité complexe que vous pouvez utiliser pour résoudre des relations complexes, notamment des relations d'auto-référencement (appelées également « fermetures transitives »). Tenez compte des instructions suivantes avant d'utiliser les déclencheurs récursifs:

- Les déclencheurs récursifs sont complexes et doivent être conçus dans les règles de l'art et avec minutie. Les déclencheurs récursifs exigent un code de bouclage logique (contrôle d'arrêt). Sinon, vous dépasserez la limite d'imbrication de 32 niveaux.
- Une modification de données à un moment donné peut activer la série de déclencheurs. Bien que cette situation offre la possibilité de traiter des relations complexes, elle peut poser un problème si vos tables sont à mettre à jour dans un ordre précis.
- Vous pouvez créer des fonctionnalités semblables sans la fonction de déclencheur récursif mais la structure de votre déclencheur sera légèrement différente. Lorsque vous concevez des déclencheurs récursifs, chacun de vos déclencheurs doit inclure un contrôle conditionnel chargé d'arrêter le traitement récursif si la condition prend la valeur False. La création de déclencheurs non récursifs doit intégrer les structures de bouclage et les contrôles complets des langages de programmation.

# 8. Application pratique : Création de déclencheurs

## 8.1. Suppression de contraintes existantes

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes** et **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Dans le menu **Fichier**, pointez sur **Ouvrir**, puis cliquez sur **Fichier**.
4. Accédez au dossier **D:\Practices**, puis ouvrez le fichier **DropTriggers.sql**.
5. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
6. Fermez la requête **DropTriggers.sql** .

## 6.1. Créer un déclencheur UPDATE

Vous devez exécuter les étapes suivantes pour créer un déclencheur UPDATE à l'aide de Transact-SQL:

1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** de la barre d'outils.
2. Dans l'Explorateur d'objets, développez **Bases de données**, **AdventureWorks**, **Tables**, **HumanResources.Employee**, puis cliquez sur **Déclencheurs**.
3. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks
GO
CREATE TRIGGER uEmployee ON HumanResources.Employee AFTER UPDATE NOT FOR
REPLICATION AS
BEGIN

    SET NOCOUNT ON;
    UPDATE HumanResources.Employee
    SET HumanResources.Employee.ModifiedDate = GETDATE() FROM inserted WHERE inserted.EmployeeID
    HumanResources.Employee.EmployeeID
END
```

4. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
5. Une fois l'exécution réussie, cliquez avec le bouton droit sur le dossier **Déclencheurs** dans l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que le déclencheur **uEmployee** a été créé.
6. Cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
7. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks
GO
UPDATE HumanResources . Employee
SET ModifiedDate = '01/01/01'
WHERE EmployeeID = 1

SELECT ModifiedDate
FROM HumanResources. Employee
WHERE EmployeeID = 1
```

8. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
9. Vérifiez que le déclencheur a défini la valeur **ModifiedDate** sur la valeur de date et heure actuelle, même si vous l'avez mise à jour au 1er janvier 2001.

## 6.2. Créer un déclencheur INSTEAD OF

Vous devez exécuter les étapes suivantes pour créer un déclencheur INSTEAD OF à l'aide de Transact-SQL:

1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** de la barre d'outils.
2. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks GO
CREATE TRIGGER dEmployee ON HumanResources.Employee
INSTEAD OF DELETE NOT FOR REPLICATION AS
BEGIN
    SET NOCOUNT ON

    DECLARE @DeleteCount int

    SELECT @DeleteCount = COUNT(*) FROM deleted
    IF @DeleteCount > 0
    BEGIN
        RAISERROR
            (‘Employees cannot be deleted. They can only be marked as not current.', 10,
            1)
        -- Roll back any active or uncommittable transactions
        IF @@TRANCOUNT > 0
        B EGIN
            ROLLBACK TRANSACTION;
        END
      END
   END
```

3. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
4. Une fois l'exécution réussie, cliquez avec le bouton droit sur le dossier **Déclencheurs** dans l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que le déclencheur **dEmployee** a été créé.
5. Cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
6. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks
GO
DELETE FROM HumanResources.Employee WHERE EmployeeID = 1
```

7. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
8. Vérifiez que la suppression échoue et que le déclencheur retourne une erreur.

# Atelier pratique : Implémentation de l’intégrité des données

## 1. Scénario

Le service des ressources humaines souhaite conserver un enregistrement historique des candidats postulant aux postes publiés pour être en mesure de recontacter des candidats recalés si de nouveaux postes sont créés et de comparer les informations fournies par les postulants s'ils soumettent leur candidature pour différents postes à venir. Après s'être concerté avec l'équipe du service des ressources humaines, le développeur en chef de la base de données estime que l'utilisation d'une nouvelle table capable de stocker ces données historiques est nécessaire. Il vous confie alors le soin d'effectuer les opérations suivantes:

- Créer une nouvelle table nommée **HumanResourcesjobCandidateHistory**. La table

**JobCandidateHistory** devra contenir les colonnes et contraintes suivantes:

- ✓ **JobCandidateID.** Colonne **int** ne pouvant contenir aucune valeur NULL. Les valeurs de cette colonne doivent être uniques.
- ✓ **Resume.** Colonne **xml** pouvant contenir des valeurs NULL.
- ✓ **Rating.** Colonne **int** ne pouvant contenir aucune valeur NULL. Les valeurs de cette colonne doivent s'inscrire dans la plage 1 à 10, avec une valeur par défaut de 5.
- ✓ **RejectedDate.** Colonne **datetime** ne pouvant contenir aucune valeur NULL.
- ✓ **ContactID.** Colonne **int** pouvant contenir des valeurs NULL. Cette colonne est une clé étrangère de la colonne **ContactID** dans la table **Person.Contact**.

- Créer un nouveau déclencheur DELETE nommé **dJobCandidate** dans la table **HumanResourcesjobCandidate** chargé de copier les informations du postulant dans la table **HumanResourcesjobCandidateHistory** lorsque quelqu'un supprime un postulant. Vous devez copier les colonnes **JobCandidateId** et **Resume** directement et définir **RejectedDate** sur la date actuelle à l'aide de la fonction **getdate**. **Rating** doit être conservé en tant que valeur par défaut et **ContactDetails** doit être défini sur NULL.

- Créer un projet de script SQL Server pour les modifications à l'aide de SQL Server Management Studio, puis stocker le projet dans le dossier D:\Labfiles\Starter.

## 2. Exercice 1 : Création de contraintes

| | |
|---|---|
| Créer la table **JobCandidateHistory** et les contraintes. | 1. Dans la fenêtre de requête, tapez l'instruction Transact-SQL appropriée pour créer la table **HumanResources.JobCandidateHistory** dans la base de données **AdventureWorks**.<br>2. Exécutez la requête, puis enregistrez le fichier de requête.<br>3. Utilisez l'Explorateur d'objets pour vérifier que la table et les contraintes ont été créées.<br>4. Laissez SQL Server Management Studio ouvert. Vous allez l'utiliser au cours de l'exercice suivant. |
| Tester la table **JobCandidateHistory** et les contraintes. | 1. Dans SQL Server Management Studio, ouvrez le fichier de script **TestConstraints.sql** dans le dossier D:\Labfiles\Starter.<br>2. Sélectionnez le code sous le commentaire **Cette opération devrait échouer**, puis cliquez sur le bouton **Exécuter**. L'opération INSERT devrait se solder par un échec puisque la valeur **Rating** est en conflit avec la contrainte CHECK.<br>3. Sélectionnez le code sous le commentaire **Cette opération devrait réussir**, puis cliquez sur le bouton **Exécuter**. L'opération INSERT devrait réussir. |

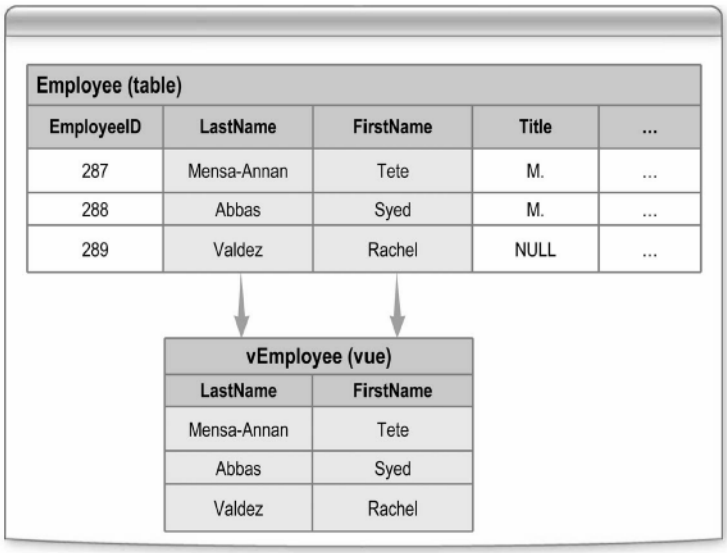
## 3. Exercice 2 : Création de déclencheurs

| | |
|---|---|
| Créer le déclencheur **dJobCandidate**. | 1. Dans la fenêtre de requête, tapez l'instruction Transact-SQL appropriée pour créer le déclencheur **dJobCandidate** dans la table **HumanResources.JobCandidate** de la base de données **AdventureWorks**.<br>2. Exécutez la requête, puis enregistrez le fichier de requête.<br>3. Utilisez l'Explorateur d'objets pour vérifier que le déclencheur a été créé.<br>4. Laissez SQL Server Management Studio ouvert. Vous allez l'utiliser au cours de l'exercice suivant. |
| Tester le déclencheur **dJobCandidate**. | 1. Dans SQL Server Management Studio, ouvrez le fichier de script **TestTrigger.sql** dans le dossier D:\Labfiles\Starter.<br>2. Exécutez le script de la requête et examinez le volet de résultats pour vérifier que l'enregistrement supprimé dans la table **HumanResources.JobCandidate** a été inséré dans la table **HumanResources.JobCandidateHistory**. |

# MODULE 5 : IMPLEMENTATION DE VUES

## Leçon 1 : Présentation des vues

### 1. Qu'est-ce qu'une vue ?




Une vue est une table virtuelle dont le contenu est défini par une requête. Tout comme une table réelle, une vue est composée d'un ensemble de lignes et de colonnes de données nommées. À moins d'être indexée, une vue n'existe pas en tant qu'ensemble de valeurs de données stocké dans une base de données. Les lignes et les colonnes de données proviennent des tables référencées dans la requête définissant la vue et sont produites dynamiquement lorsque la vue est référencée. Les tables interrogées dans une vue répondent au nom de « tables de base ».

La liste suivante présente des exemples courants de vues:

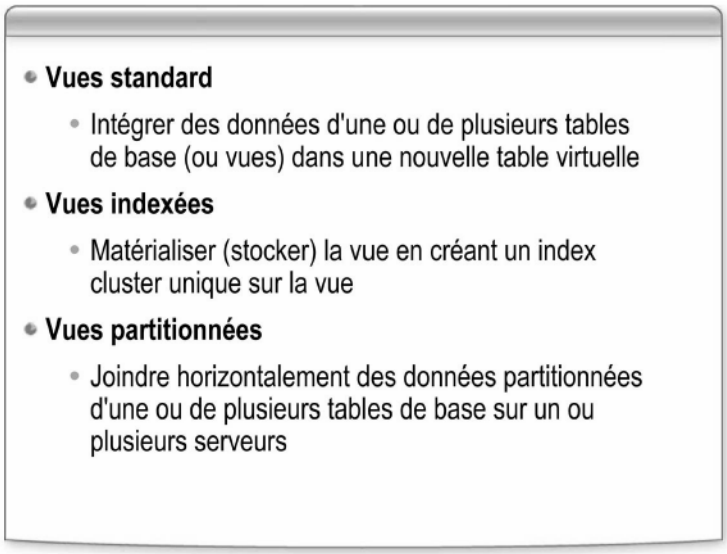
- Un sous-ensemble de lignes ou de colonnes d'une table de base
- Une union de deux tables de base ou plus
- Une jointure de deux tables de base ou plus
- Un résumé statistique d'une table de base
- Un sous-ensemble d'une autre vue ou une combinaison de vues et de tables de base

Les vues sont généralement utilisées pour:

- Permettre aux utilisateurs de se concentrer spécifiquement sur les données qui les intéressent et sur les tâches dont ils sont responsables. Les données inutiles ou sensibles peuvent être exclues de la vue.
- Simplifier la manière dont les utilisateurs manipulent les données. Vous pouvez définir des jointures, des projections, des requêtes UNION et des requêtes SELECT fréquemment utilisées en tant que vues, afin que les utilisateurs n'aient pas besoin de spécifier toutes les conditions et qualifications chaque fois qu'une nouvelle opération est effectuée sur ces données.
- Améliorer la sécurité en permettant aux utilisateurs d'accéder aux données par le biais de la vue, sans leur accorder l'autorisation d'accéder directement aux tables de base sous-jacentes de la vue.
- Assurer une compatibilité descendante en définissant une vue destinée à émuler une table qui existait mais dont le schéma a changé.

- Définir l'ensemble de données qu'un utilisateur peut importer/exporter dans SQL Server.
- Fournir une représentation consolidée des données partitionnées, c'est-à-dire des données similaires qui sont stockées dans plusieurs tables.

# 2. Types de vues




Dans SQL Server 2005, vous pouvez créer trois types de vues:

- Vues standard
- Vues indexées
- Vues partitionnées

## 2.1. Vues standards

Une vue standard intègre les données d'une ou de plusieurs tables de base dans une nouvelle table virtuelle. Les lignes d'une vue standard ne sont pas stockées ; seule la définition de la vue l'est. Chaque fois que la vue est référencée, le moteur de base de données crée les données de la vue dynamiquement. Les vues standard sont celles que vous utiliserez le plus souvent.

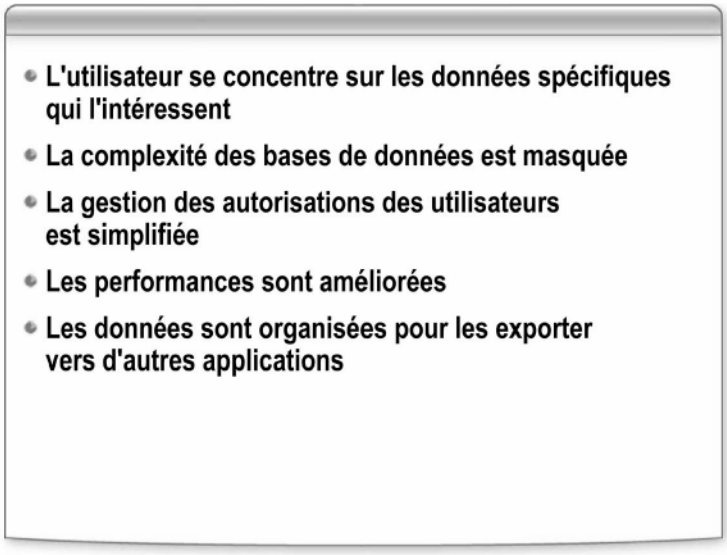
## 2.2. Vues indexées

Une vue indexée est une vue qui a été matérialisée, autrement dit calculée et stockée.
Pour indexer une vue, vous devez créer un index cluster unique sur la vue. Les vues indexées améliorent considérablement les performances de certains types de requêtes. Les vues indexées sont idéales pour les requêtes qui agrègent de nombreuses lignes, mais elles ne conviennent pas pour les tables de base fréquemment mises à jour.

## 2.3. Vues partitionnées

Une vue partitionnée joint horizontalement les données partitionnées d'une ou de plusieurs tables de base sur un ou plusieurs serveurs. Ainsi, les données similaires provenant de plusieurs tables de base sont affichées comme si elles provenaient d'une seule table. Une vue qui joint des tables membres sur la même instance de SQL Server est une vue de données partitionnées (serveur local). Lorsqu'une vue joint des données provenant de tables de différents serveurs, il s'agit d'une vue partitionnée de données distribuées.

# 3. Avantages des vues




Les vues présentent les avantages suivants:

***Elles permettent à un utilisateur de se concentrer sur les données qui l'intéressent.*** Les vues créent un environnement contrôlé qui permet d'accéder à des données spécifiques alors que d'autres données sont masquées. Les données qui sont inutiles, sensibles ou non pertinentes peuvent être exclues d'une vue. Les utilisateurs peuvent manipuler l'affichage des données dans une vue, de la même manière que dans une table. Qui plus est, avec les autorisations adaptées et en tenant compte de certaines restrictions, les utilisateurs peuvent modifier les données produites par une vue.

***Elles permettent de masquer la complexité des bases de données***. Les vues masquent aux yeux de l'utilisateur la complexité de la conception de la base de données.
De cette manière, les développeurs sont en mesure de modifier la conception sans que l'interaction de l'utilisateur avec la base de données en soit affectée. De plus, les utilisateurs disposent d'une version plus claire des données, car vous pouvez créer des vues avec des noms qui sont plus faciles à comprendre que les noms incompréhensibles souvent utilisés dans les bases de données.

Les requêtes complexes, notamment les requêtes distribuées sur des données hétérogènes, peuvent également être masquées par le biais des vues. L'utilisateur interroge la vue au lieu d'écrire la requête ou d'exécuter un script.

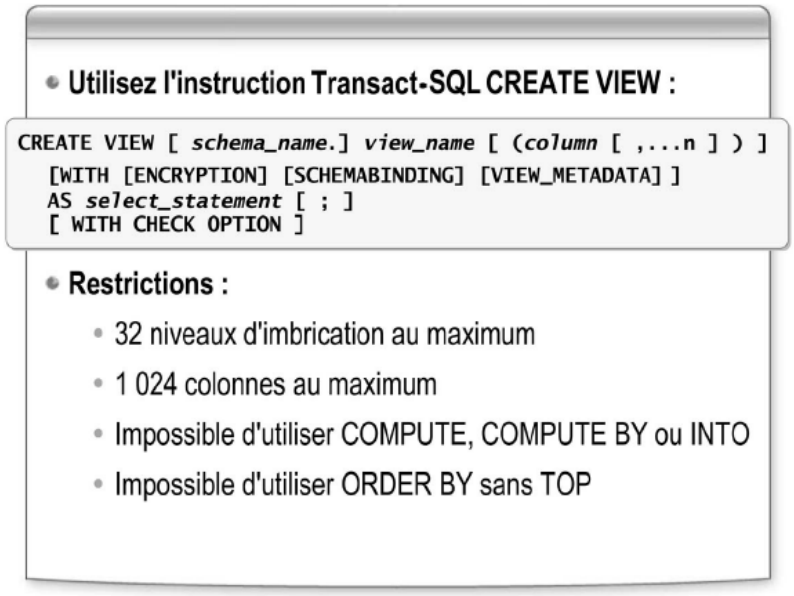
***Elles permettent de simplifier la gestion des autorisations des utilisateurs***. Au lieu d'accorder aux utilisateurs l'autorisation d'interroger des colonnes spécifiques dans les tables de base, les propriétaires de base de données peuvent accorder aux utilisateurs l'autorisation d'interroger les données uniquement par le biais des vues. Cela protège également les modifications de la conception des tables de base sous-jacentes. Les utilisateurs peuvent continuer d'interroger la vue sans interruption.

***Elles permettent d'améliorer les performances***. Les vues vous permettent de stocker les résultats des requêtes complexes. D'autres requêtes peuvent utiliser ces résultats résumés. Les vues vous permettent également de partitionner des données. Vous pouvez placer les partitions individuellement sur des ordinateurs distincts et les associer en toute transparence pour l'utilisateur.

***Elles permettent d'organiser les données pour les exporter vers d'autres applications.*** Vous pouvez créer une vue basée sur une requête complexe qui joint au moins deux tables, puis exporter les données vers une autre application pour une analyse plus approfondie.

# Leçon 2 : Création et gestion de vues

## 1. Syntaxe utilisée pour créer des vues




Vous pouvez créer des vues en exécutant l'instruction Transact-SQL CREATE VIEW ou en utilisant l'interface de conception graphique fournie dans SQL Server Management Studio. Dans le cadre de la définition d'une vue, vous pouvez spécifier le contenu de la vue à l'aide d'une instruction SELECT.

## 1.1. Création de vues

Pour créer une vue, utilisez le concepteur de vues inclus dans SQL Server Management Studio ou l'instruction Transact-SQL CREATE VIEW. Pour ouvrir le concepteur de vues dans l'Explorateur d'objets, développez la base de données que vous voulez utiliser, cliquez avec le bouton droit sur le nœud **Vues**, puis cliquez sur **Nouvelle vue**. Vous concevez votre vue à l'aide d'une interface graphique qui vous permet de sélectionner les tables et les colonnes à inclure dans la vue, de définir les relations entre les colonnes, de déterminer les lignes qui sont retournées et de configurer des options telles que les alias de colonnes et les ordres de tri utilisés pour générer la vue.

La syntaxe de l'instruction CREATE VIEW est la suivante:

```
CREATE VIEW [ schema_name . ] view_name [ (column [ ,...n ] ) ]
[ WITH [ ENCRYPTION ] [, SCHEMABINDING ] [, VIEW_METADATA ] ]
AS select_statement [ ; ]
[ WITH CHECK OPTION ]
```

Le code suivant illustre le processus de création d'une vue nommée **HumanResources.vEmployee**, lequel est composé d'un ensemble de colonnes provenant de différentes tables de la base de données **AdventureWorks**.

```
CREATE VIEW [HumanResources].[vEmployee]
AS
SELECT
    e.[EmployeeID],c.[Title],c.[FirstName],c.[MiddleName],c.[LastName]
    ,c.[Suffix],e.[Title] AS [JobTitle],c.[Phone],c.[EmailAddress]
    ,c.[EmailPromotion],a.[AddressLine1],a.[AddressLine2],a.[City]
    ,sp.[Name] AS [StateProvinceName],a.[PostalCode]
    ,cr.[Name] AS [CountryRegionName],c.[AdditionalContactInfo]
FROM [HumanResources].[Employee] e
    INNER JOIN [Person].[Contact] c
    ON c.[ContactID] = e.[ContactID]
    INNER JOIN [HumanResources].[EmployeeAddress] ea
    ON e.[EmployeeID] = ea.[EmployeeID]
    INNER JOIN [Person].[Address] a
    ON ea.[AddressID] = a.[AddressID]
    INNER JOIN [Person].[StateProvince] sp
    ON sp.[StateProvinceID] = a.[StateProvinceID]
    INNER JOIN [Person].[CountryRegion] cr
    ON cr.[CountryRegionCode] = sp.[CountryRegionCode]
```

## 1.2. Impératifs liés à la création de vues

Lorsque vous créez des vues, tenez compte des impératifs suivants:

- Vous devez être membre du rôle **sysadmin**, du rôle **db_owner** ou du rôle **db_ddladmin**, ou vous devez bénéficier de l'autorisation CREATE VIEW dans la base de données et de l'autorisation ALTER SCHEMA sur le schéma dans lequel la vue doit être créée. Vous devez également bénéficier de l'autorisation SELECT sur toutes les tables ou les vues référencées dans la vue.
- Vous pouvez créer des vues uniquement dans la base de données active.
- Le nom de votre vue doit respecter les règles en vigueur pour les identificateurs et doit être différent de tous les autres noms de table ou de vue dans la base de données.
- Vous pouvez créer des vues sur d'autres vues. L'imbrication ne doit pas excéder 32 niveaux, mais elle peut également être limitée par la complexité des vues et la mémoire disponible.
- Votre vue ne peut pas contenir plus de 1 024 colonnes.
- Vous devez spécifier des noms de colonnes si:
  - ✓ une ou plusieurs colonnes de la vue sont dérivées d'une expression arithmétique, d'une fonction intégrée ou d'une constante;
  - ✓ certaines colonnes dans les tables qui seront jointes partagent le même nom.
- Vous ne pouvez ni créer des vues temporaires, ni créer des vues à partir de tables temporaires.
- Vous ne pouvez pas associer des objets **de règle** ou des objets **par défaut** à une vue.
- Vous ne pouvez pas associer des déclencheurs AFTER à des vues, mais seulement des déclencheurs INSTEAD OF.

# 2. Les vues avec SQL Server Management Studio

## 2.1. Création de vues

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, sur **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Dans l'Explorateur d'objets, développez **Bases de données**, **AdventureWorks**, puis **Vues**.
4. Cliquez avec le bouton droit sur **Vues**, puis cliquez sur **Nouvelle vue**.
5. Sélectionnez les tables **Contact (Person)** et **Employee (HumanResources)** dans la boîte de dialogue **Ajouter une table** (maintenez la touche Ctrl enfoncée pour sélectionner les deux tables), cliquez sur **Ajouter**, puis cliquez sur **Fermer**.
6. Sélectionnez **EmployeeID** et **Title** dans la table **Employee (HumanResources)**.
7. Sélectionnez **Title**, **FirstName**, **MiddleName**, **LastName**, **EmailAddress** et **Phone** dans la table **Contact (Person)**.
8. Changez l'**alias** de la colonne **Person.Contact.Title** en remplaçant **Expr1** par **Salutation**.
9. Si la fenêtre Propriétés n'est pas visible, cliquez sur **Fenêtre Propriétés** dans le menu **Affichage**.
10. Attribuez à la propriété **Schéma** de la vue la valeur **HumanResources**.
11. Dans le menu **Fichier**, cliquez sur **Enregistrer Vue – dbo.View_1**.
12. Entrez le nom **vEmployeeContactView** pour la nouvelle vue, puis cliquez sur **OK**.

## 2.2. Interrogation de vues

Pour afficher le contenu de la vue **vEmployeeContactView** à l'aide de SQL Server Management Studio:

1. Dans SQL Server Management Studio, cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE AdventureWorks
GO
SELECT * FROM
HumanResources.vEmployeeContactView
```

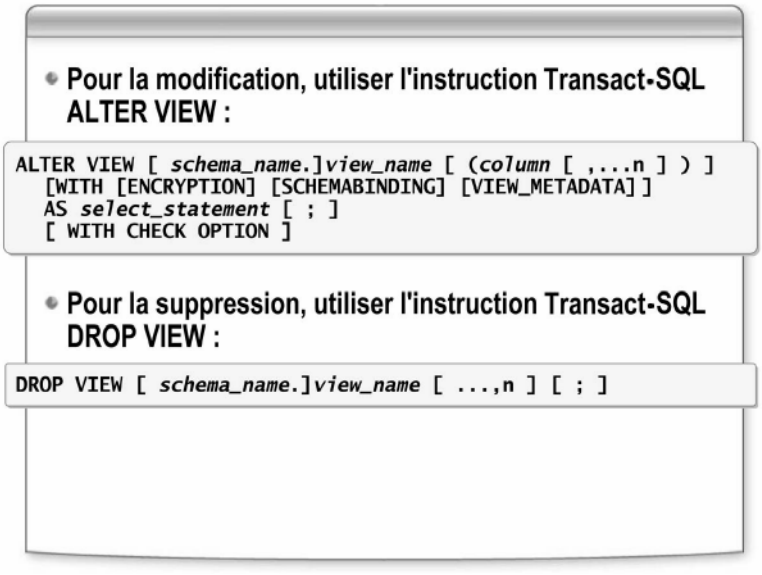
4. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
5. Une fois la commande exécutée, affichez les résultats retournés.
6. Ne fermez pas SQL Server Management Studio, car vous allez l'utiliser dans la procédure suivante.

## 2.3. Génération d'un script d'une vue

Pour afficher la syntaxe Transact-SQL utilisée pour créer la vue **vEmployeeContactView**:

1. Dans l'Explorateur d'objets, cliquez avec le bouton droit sur le nœud **Vues** sous la base de données **AdventureWorks**, puis cliquez sur **Actualiser**.
2. Cliquez avec le bouton droit sur la vue **HumanResources.vEmployeeContactView**, cliquez sur **Générer un script de la vue en tant que**, cliquez sur **CREATE To**, puis cliquez sur **Nouvelle fenêtre d'éditeur de requête**.
3. Passez en revue l'instruction Transact-SQL CREATE VIEW.

# 3. Syntaxe utilisée pour modifier et supprimer des vues




Si vous devez modifier une vue, vous pouvez soit recourir à SQL Server Management Studio, soit exécuter l'instruction Transact-SQL ALTER VIEW.
Si une vue est devenue inutile, vous pouvez supprimer sa définition de la base de données en utilisant SQL Server Management Studio ou en exécutant l'instruction Transact-SQL **DROP VIEW**.

## 3.1. Modification des vues

Vous pouvez modifier la définition de votre vue en ouvrant l'outil concepteur de vues dans l'Explorateur d'objets ou en utilisant l'instruction Transact-SQL ALTER VIEW. Vous pouvez modifier les tables et les colonnes incluses dans la vue, modifier les relations entre les colonnes, restreindre les lignes retournées par la vue et modifier des options telles que les alias de colonnes et les ordres de tri utilisés pour générer la vue.

L'instruction ALTER VIEW modifie la définition d'une vue, y compris les vues indexées, sans aucune incidence sur les déclencheurs ou les procédures stockées. Vous pouvez ainsi conserver les autorisations de la vue. Cette instruction est soumise aux mêmes restrictions que l'instruction CREATE VIEW. La syntaxe de l'instruction ALTER VIEW est la suivante:

```
ALTER VIEW [ schema_name . ] view_name [ ( column [ ,...n ] ) ]
[ WITH <view_attribute> [ ,...n ] ]
AS select_statement [ ; ]
[ WITH CHECK OPTION ]

<view_attribute> ::=
{
      [ ENCRYPTION ]
      [ SCHEMABINDING ]
      [ VIEW_METADATA ]
}
```

Le code suivant modifie la vue **HumanResources.vEmployee** de la table **AdventureWorks** et supprime de la vue toutes les informations concernant l'adresse postale.

```
ALTER VIEW [HumanResources].[vEmployee]
AS
SELECT
    e.[EmployeeID]
    ,c.[Title]
    ,c.[FirstName]
    ,c.[MiddleName]
    ,c.[LastName]
    ,c.[Suffix]
    ,e.[Title] AS [JobTitle]
    ,c.[Phone]
    ,c.[EmailAddress]
FROM [HumanResources].[Employee] e
    INNER JOIN [Person].[Contact] c
    ON c.[ContactID] = e.[ContactID]
```

## 3.2.Suppression des vues

La suppression d'une vue se traduit par la suppression de sa définition et de toutes les autorisations qui lui sont affectées. Par ailleurs, si un utilisateur interroge une vue qui référence la vue supprimée, il reçoit alors un message d'erreur. Cependant, la suppression d'une table qui est référencée par une vue n'entraîne pas la suppression automatique de la vue. Vous devez supprimer la vue explicitement.

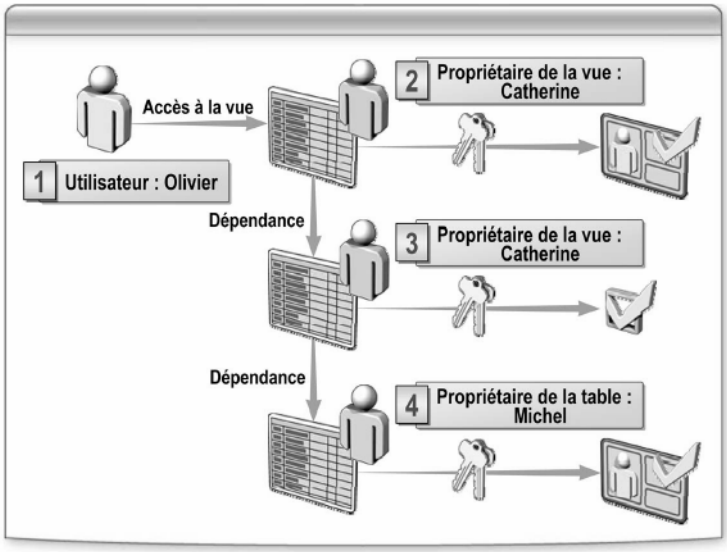
Pour procéder à la suppression d'une vue, vous pouvez soit supprimer la vue dans l'Explorateur d'objets, soit utiliser l'instruction DROP VIEW. La syntaxe de l'instruction DROP VIEW est la suivante:

```
DROP VIEW [ schema_name . ] view_name [ . . . ,n ] [ ; ]
```

Le code suivant présente l'instruction utilisée pour supprimer la vue **HumanResources.vEmployee** de la base de données **AdventureWorks**.

```
DROP VIEW [HumanResources].[vEmployee]
```

# 4. Influence des chaînes de propriétés sur les vues




Les vues dépendent des vues ou des tables sur lesquelles elles reposent. Dans la mesure où SQL Server résout le contenu d'une vue, il parcourt la hiérarchie des dépendances de tables et d'affichage pour récupérer le contenu approprié. Lorsque plusieurs objets de base de données accèdent les uns aux autres de manière séquentielle, la séquence est désignée sous le nom de « chaîne ». SQL Server n'évalue pas les autorisations sur les objets dans une chaîne de la même manière que si l'accès aux objets s'opérait de manière indépendante.

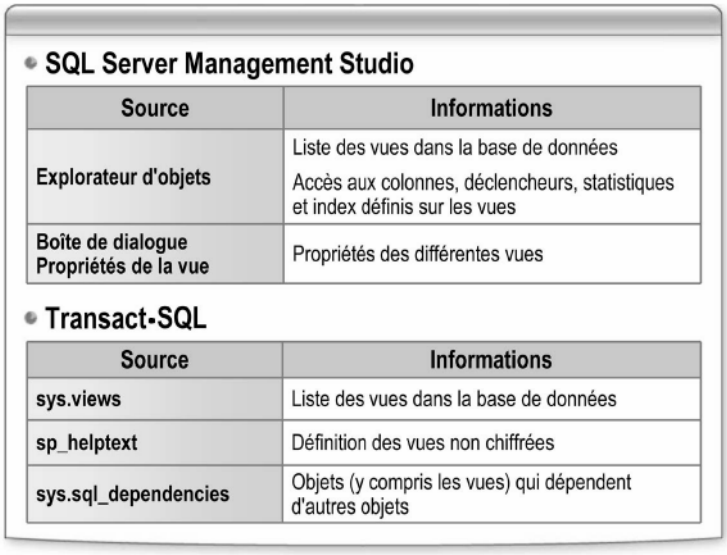
## 4.1. Vérification des autorisations par SQL Server dans une chaîne de propriétés

Lorsqu'un utilisateur accède à un objet de base de données (tel qu'une vue) et que celui- ci accède à un autre objet (tel qu'une table), SQL Server ne vérifie pas les autorisations de l'utilisateur sur le second objet si les deux objets appartiennent au même propriétaire. En d'autres termes, lorsqu'un utilisateur est autorisé à accéder à une vue, SQL Server n'évalue pas les autorisations de l'utilisateur sur chacune des vues ou tables sous-jacentes dès lors que ces dernières appartiennent au propriétaire de la vue d'origine. Lorsqu'un utilisateur accède à un objet qui n'a pas le même propriétaire, SQL Server évalue, dans ce cas uniquement, les autorisations de l'utilisateur sur cet objet.

Le chaînage des propriétés vous permet de gérer l'accès à plusieurs objets (tels que des tables) en affectant des autorisations à un objet (tel qu'une vue). Le chaînage des propriétés offre également un léger avantage en termes de performances dans les scénarios qui permettent d'ignorer la vérification des autorisations.

Pour éviter la rupture des chaînes de propriétés, vérifiez que toutes les vues ainsi que les fonctions et les tables sous-jacentes ont le même propriétaire.

# 5. Source d'informations concernant les vues



- **SQL Server Management Studio**

| Source | Informations |
|---|---|
| Explorateur d'objets | Liste des vues dans la base de données<br>Accès aux colonnes, déclencheurs, statistiques et index définis sur les vues |
| Boîte de dialogue Propriétés de la vue | Propriétés des différentes vues |

- **Transact-SQL**

| Source | Informations |
|---|---|
| sys.views | Liste des vues dans la base de données |
| sp_helptext | Définition des vues non chiffrées |
| sys.sql_dependencies | Objets (y compris les vues) qui dépendent d'autres objets |

Vous pourrez avoir besoin d'informations sur les vues existantes avant de créer, demodifier ou de supprimer une vue. Dans SQL Server 2005, vous pouvez utiliser les sources suivantes pour obtenir des informations sur les vues:

- SQL Server Management Studio
- Affichage catalogue **sys.views**
  - Procédure stockée système **sp_helptext**
  - Affichage catalogue **sys.sql_dependencies**

## 5.1. Obtention d'informations via SQL Server Management Studio

Pour obtenir des informations sur une vue à l'aide de SQL Server Management Studio, ouvrez l'Explorateur d'objets et développez la base de données que vous souhaitez utiliser. Développez le nœud Vues pour consulter la liste des vues disponibles.
Développez une vue pour accéder aux colonnes, aux déclencheurs, aux index et aux statistiques définis pour la vue. Vous pouvez aussi cliquer avec le bouton droit pour afficher un menu contextuel vous permettant de générer les scripts utilisés pour créer, modifier et supprimer la vue, et d'ouvrir la boîte de dialogue Propriétés de la vue.

## 5.2. Obtention d'une liste de vues via Transact-SQL

Vous pouvez interroger les affichages catalogue **sys.views** pour obtenir des informations d'ordre général sur les vues disponibles. L'ensemble de résultats contiendra une ligne pour chaque vue disponible. L'exemple suivant retourne la liste de toutes les vues disponibles dans la base de données **AdventureWorks**.

```
USE AdventureWorks
GO
SELECT * FROM sys.views
```

## 5.3. Obtention de la définition d'une vue via Transact-SQL

Pour afficher la définition d'une vue non chiffrée, utilisez la procédure stockée système **sp_helptext**, en transmettant le nom de la vue en tant qu'argument. La procédure stockée **sp_helptext** retourne une erreur si la vue spécifiée est chiffrée. L'exemple suivant retourne la définition de la vue **HumanResources.vEmployee** de la base de données **AdventureWorks**.

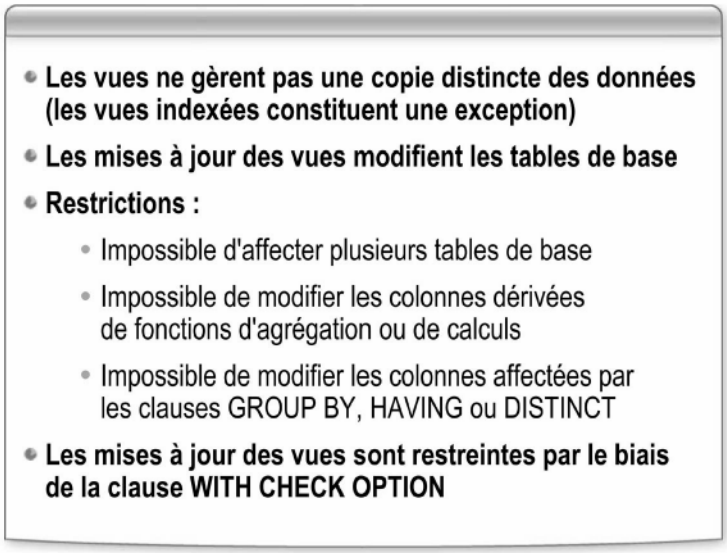
```
USE AdventureWorks
GO
EXEC sp_helptext 'HumanResources.vEmployee'
```

### 5.4.Détermination des dépendances d'affichage via Transact-SQL

Avant de supprimer une table ou une vue, utilisez l'affichage catalogue **sys.sql_dependencies** pour déterminer si d'autres vues dépendent de cette table ou vue. L'affichage **sys.sql_dependencies** contient une ligne pour chaque dépendance avec une table ou vue référencée. L'exemple suivant indique comment obtenir la liste de tous les objets de base de données qui ont des dépendances directes avec la table **HumanResources.Employee** de la base de données **AdventureWorks**.

```
USE AdventureWorks
GO
SELECT DISTINCT OBJECT_NAME(object_id) AS Name
FROM sys.sql_dependencies
WHERE referenced_major_id =
OBJECT_ID(N'AdventureWorks.HumanResources.Employee')
```

## 6. Eléments à prendre à compte pour la modification des données dans une vue




Les vues ne gèrent pas une copie distincte des données. Au lieu de cela, elles affichent l'ensemble de résultats d'une requête exécutée sur une ou plusieurs tables de base.
Par conséquent, chaque fois que vous modifiez des données dans une vue, vous modifiez en fait la table de base. En tenant compte de certaines restrictions, vous pouvez insérer, mettre à jour ou supprimer librement des données de table par l'intermédiaire d'une vue.

### 6.1.Restrictions s'appliquant à la modification des données dans une vue

Les restrictions suivantes s'appliquent lorsque vous tentez de modifier des données dans une vue:

- Toutes les modifications, y compris celles opérées via les instructions UPDATE, INSERT et DELETE, doivent référencer les colonnes d'une seule et même table de base.
- Toutes les colonnes qui font l'objet de modifications doivent référencer directement les données sous-jacentes figurant dans les colonnes des tables. Elles ne peuvent être dérivées d'aucune autre manière (par exemple, à partir d'une fonction d'agrégation ou d'un calcul qui utilise d'autres colonnes).

- Les colonnes qui font l'objet de modifications ne peuvent pas être affectées par les clauses GROUP BY, HAVING ou DISTINCT.

### 6.2. Utilisation de CHECK OPTION pour imposer la vérification des conditions lors des mises à jour

Toutes les instructions de modification de données exécutées sur la vue doivent respecter les critères définis dans l'instruction SELECT qui définit la vue, si la clause WITH CHECK OPTION est utilisée dans la définition de la vue. Si vous incluez la clause WITH CHECK OPTION, les lignes ne peuvent pas être modifiées de telle façon qu'elles disparaissent de la vue. Toute tentative de modification d'une ligne pouvant entraîner cette situation est annulée et provoque l'affichage d'un message d'erreur.

# 7. Application pratique : Création d'une vue

## 7.1. Création d'une vue à l'aide de Transact-SQL

Vous devez exécuter les étapes suivantes pour créer une vue à l'aide de Transact-SQL:

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, sur **Microsoft SQL Server 2005**, puis cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, spécifiez les valeurs dans le tableau suivant, puis cliquez sur **Se connecter**.
3. Si l'Explorateur d'objets n'est pas visible, cliquez sur **Explorateur d'objets** dans le menu **Affichage**.
4. Dans l'Explorateur d'objets, développez **Bases de données**, **AdventureWorks**, puis **Vues**.
5. Dans le menu **Fichier**, pointez sur **Ouvrir**, puis cliquez sur **Fichier**.
6. Accédez au dossier **D:\Practices** et ouvrez le fichier **CreateView.sql**.
7. Vérifiez le code suivant dans la fenêtre de la requête.

```
USE AdventureWorks GO
CREATE VIEW [HumanResources].[vEmployeeContact]
WITH ENCRYPTION AS SELECT
   e.EmployeeID, e.Title,
   c.Title AS Salutation,
   c.FirstName,c.MiddleName, c.LastName,
   c. EmailAddress, c.Phone
FROM Person.Contact c INNER JOIN HumanResources.Employee e
    ON c.ContactID = e.ContactID
```

8. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
9. Une fois la commande exécutée, cliquez avec le bouton droit sur le dossier **Vues** dans l'Explorateur d'objets, puis cliquez sur **Actualiser** pour vérifier que la vue **HumanResources.vEmployeeContact** a été créée.
10. Cliquez sur le bouton **Nouvelle requête** dans la barre d'outils.
11. Dans la nouvelle fenêtre de requête vide, tapez le code Transact-SQL suivant.

```
USE [AdventureWorks]
GO
SELECT * FROM
[HumanResources].[vEmployeeContact]
```

12. Cliquez sur le bouton **Exécuter** dans la barre d'outils.
13. Vérifiez que les données sont retournées à partir de la vue.

5. Fermez SQL Server Management Studio. Cliquez sur **Non** lorsque vous êtes invité à enregistrer les fichiers.

# Leçon 3 : Optimisation des performances à l'aide de vues

## 1. Eléments à prendre en compte en matière de performances pour les vues

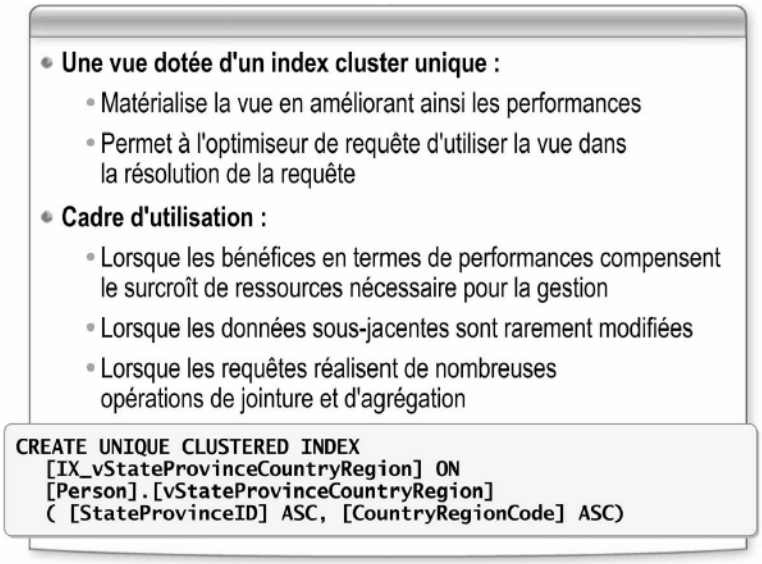
L'importante sollicitation des ressources système est inhérente à l'utilisation d'une vue standard, car SQL Server doit exécuter des instructions SELECT sur une ou plusieurs tables pour récupérer les données de la vue lors de chaque accès à cette dernière. Cette surcharge qui affecte les performances est accrue si les vues sont imbriquées. Lorsque vous imbriquez des vues, veillez à ne pas introduire par inadvertance une longue chaîne de vues imbriquées qui affectera considérablement les performances des requêtes.
Pour savoir si les vues sont imbriquées, il vous suffit d'observer la définition de la vue afin de déterminer si elle repose sur des tables ou des vues. Toutefois, lorsqu'une vue est chiffrée, il est impossible d'examiner sa définition. Pour évaluer les performances d'une vue et identifier les actions exécutées par une vue imbriquée qui est chiffrée, vous pouvez utiliser le Générateur de profils SQL Server.

### 1.1. Amélioration des performances des vues

Pour améliorer les performances des vues, deux approches sont généralement adoptées:

- Vues indexées
- Vues partitionnées

## 2. Qu'est-ce qu'une vue indexée ?




Une vue indexée est une vue qui comporte un index cluster unique, créé sur cette dernière. Une vue indexée stocke l'ensemble de résultats d'une vue dans les pages de niveau feuille de l'index. SQL Server peut référencer rapidement l'index pour obtenir les données de la vue

### 2.1. Avantages des vues indexées en termes de performances

Au fur et à mesure que des modifications sont apportées aux données des tables de base, ces modifications sont répercutées dynamiquement sur les données stockées dans la vue indexée. Grâce à la contrainte d'unicité de l'index cluster de la vue, SQL Server trouve plus facilement les lignes de l'index qui sont affectées par une modification de données. Étant donné que la récupération est opérée plus rapidement, vous pouvez utiliser des vues indexées pour améliorer les performances des requêtes.

La création d'un index sur une vue présente un autre avantage : l'optimiseur de requête utilise d'abord cet index dans les requêtes qui ne nomment pas directement la vue dans la clause FROM. Les requêtes existantes n'ont pas besoin d'être réécrites pour tirer parti de l'amélioration des performances obtenue lors de l'extraction des données à partir de la vue indexée.

## 2.2. Eléments à prendre en compte pour l'utilisation des vues indexées

Les vues indexées nécessitent des ressources supplémentaires pour la gestion de l'index, ce qui constitue un inconvénient. L'utilisation d'une vue indexée doit être envisagée dans les cas de figure suivants:

- lorsque les bénéfices obtenus en termes de performances des requêtes compensent le surcroît de ressources nécessaire pour la gestion de l'index;
- lorsque les données sous-jacentes sont rarement mises à jour;
- lorsque les requêtes réalisent de nombreuses opérations de jointure et d'agrégation qui traitent de nombreuses lignes ou qui sont fréquemment exécutées par un grand nombre d'utilisateurs.

## 2.3. Impératifs liés à la création de vues indexées

La création de vues indexées est soumise à un grand nombre d'impératifs et de limitations. Les principaux impératifs sont les suivants:

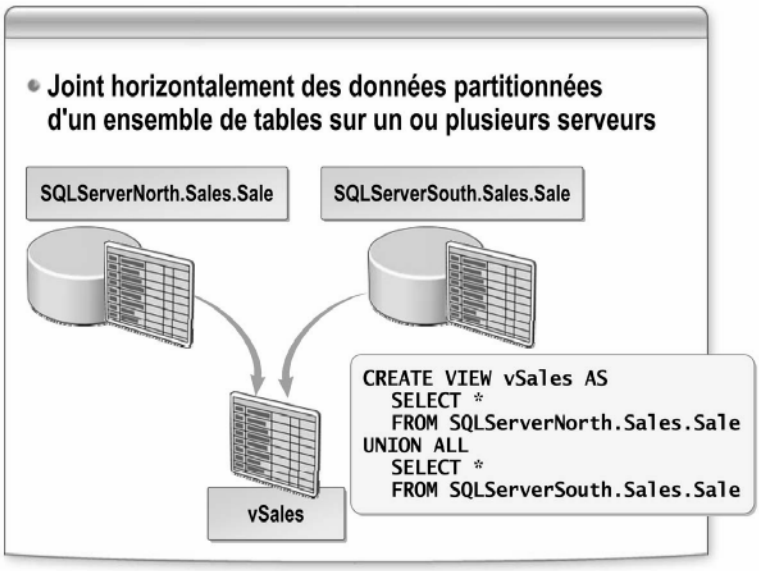
- Le premier index que vous créez sur une vue doit être un index cluster unique.
- La vue doit être définie par le biais de l'option SCHEMABINDING. Cette option de liaison de schéma lie la vue au schéma des tables de base sous-jacentes.
- La vue peut référencer des tables de base mais ne peut pas référencer d'autres vues.
- Toutes les tables de base référencées par la vue doivent figurer dans la même base de données que cette dernière et appartenir au même propriétaire.
- Les tables et les fonctions définies par l'utilisateur qui sont référencées par la vue doivent être désignées par des noms à deux composantes dans la vue. Les noms à une, trois et quatre composantes ne sont pas autorisés.
- Les fonctions référencées par des expressions dans la vue doivent être déterministes.

## 2.4. Exemple d'une vue indexée

Vous créez des index sur des vues à l'aide de l'instruction CREATE INDEX. L'exemple suivant crée un index cluster unique nommé **IX_vStateProvinceCountryRegion** sur la vue **Person.vStateProvinceCountryRegion** de la base de données **AdventureWorks**.

```
USE [AdventureWorks]
GO
CREATE UNIQUE CLUSTERED INDEX [IX_vStateProvinceCountryRegion] ON
[Person].[vStateProvinceCountryRegion]
(
	[StateProvinceID] ASC, [CountryRegionCode] ASC
)
```

# 3. Qu'est-ce qu'une vue partitionnée ?




Une vue partitionnée joint horizontalement des données partitionnées d'un ensemble de tables membres sur un ou plusieurs serveurs, les données semblant alors provenir d'une seule table. La vue partitionnée utilise des clauses UNION ALL pour intégrer les résultats des instructions SELECT exécutées sur toutes les tables membres dans un ensemble de résultats unique.

### 3.1. Types de vues partitionnées

SQL Server 2005 fait la distinction entre les vues de données partitionnées (serveur local) et les vues partitionnées de données distribuées. Dans une vue de données partitionnées (serveur local), toutes les tables participantes et la vue résident sur la même instance de SQL Server. Les vues de données partitionnées (serveur local) sont incluses dans SQL Server 2005 uniquement pour des raisons de compatibilité descendante. La méthode privilégiée pour partitionner des données localement consiste à utiliser des tables partitionnées.

Dans une vue partitionnée de données distribuées, l'une des tables participantes au minimum réside sur un serveur différent (distant). En outre, SQL Server 2005 différencie les vues partitionnées pouvant être mises à jour de celles qui sont des copies en lecture seule des tables sous-jacentes.

### 3.2. Avantages des vues partitionnées

Si les tables d'une vue partitionnée ne résident pas sur le même serveur ou résident sur un ordinateur équipé de plusieurs processeurs, chaque table impliquée dans la requête peut être analysée en parallèle, ce qui contribue à l'amélioration des performances. En outre, les tâches de maintenance, telles la reconstruction des index ou la sauvegarde d'une table, peuvent être exécutées plus rapidement.

# Atelier pratique : Implémentation de vues

## 1. Scénario

Suite à son entretien avec le service des ressources humaines, le responsable du développement de bases de données a identifié plusieurs vues qui optimiseront l'utilisation des bases de données et simplifieront les tâches de développement ultérieures. Le responsable vous a demandé d'accomplir les tâches suivantes:

- Créez une nouvelle vue nommée **HumanResources.vEmployeeDetails** qui utilise la logique SELECT suivante. Pensez à utiliser l'option SCHEMABINDING lorsque vous créez la vue, car vous devrez par la suite créer un index sur la vue.

```
SELECT
    e.[EmployeeID]
    ,c.[Title]
    ,c.[FirstName]
    ,c.[MiddleName]
    ,c.[LastName]
    ,c.[Suffix]
    ,e.[Title] AS [JobTitle]
    ,c.[Phone]
    ,c.[EmailAddress]
    ,c.[EmailPromotion]
    ,a.[AddressLine1]
    ,a.[AddressLine2]
    ,a.[City]
    ,sp.[Name] AS [StateProvinceName]
    ,a.[PostalCode]
    ,cr.[Name] AS [CountryRegionName]
    ,c.[AdditionalContactInfo]
FROM [HumanResources].[Employee] e
    INNER JOIN [Person].[Contact] c
    ON c.[ContactID] = e.[ContactID]
    INNER JOIN [HumanResources].[EmployeeAddress] ea
    ON e.[EmployeeID] = ea.[EmployeeID]
    INNER JOIN [Person].[Address] a
    ON ea.[AddressID] = a.[AddressID]
    INNER JOIN [Person].[StateProvince] sp
    ON sp.[StateProvinceID] = a.[StateProvinceID]
    INNER JOIN [Person].[CountryRegion] cr
    ON cr.[CountryRegionCode] = sp.[CountryRegionCode]
```

- Après avoir testé la vue **HumanResources.vEmployeeDetails**, transformez-la en vue indexée en créant un index cluster unique nommé **IX_vEmployeeDetails** sur la colonne **EmployeeID**.
- Créez une vue partitionnée de données distribuées nommée **Person.vContact** sur la base de données **AW_Contacts**. La vue doit associer les ensembles de données contenus dans les tables et serveurs sources suivants:
  - ✓ **[MIAMI].AW_Contacts.Person.Contact**
  - ✓ **[MIAMI\SQLINSTANCE2].AW_Contacts.Person.Contact**
  - ✓ **[MIAMI\SQLINSTANCE3].AW_Contacts.Person.Contact**

(où MIAMI est le nom du serveur)

## 2. Exercice 1 : Création de vues

| Tâche | Informations |
|---|---|
| Créer un projet de scripts SQL Server. | 1. Créez un nouveau projet de scripts SQL Server nommé **AW_Views** dans le dossier approprié.<br>2. Ajoutez une nouvelle requête au projet, en vous connectant au serveur MIAMI à l'aide de l'authentification Windows lorsque vous y êtes invité.<br>3. Remplacez le nom du fichier de requête par **CreateEmployeeView.sql**. |
| Créer la vue **HumanResources.vEmployeeDetails**. | 1. Dans la fenêtre de la requête, tapez l'instruction Transact-SQL appropriée pour créer la vue **HumanResources.vEmployeeDetails** dans la base de données **AdventureWorks**.<br>2. Exécutez la requête, puis enregistrez le fichier de requête.<br>3. Utilisez l'Explorateur d'objets pour vérifier que la vue a été créée. |
| Tester la vue **HumanResources.vEmployeeDetails**. | 1. Créez une nouvelle requête contenant le code suivant :<br>`Use AdventureWorks`<br>`SELECT * FROM`<br>`[HumanResources].[vEmployeeDetails]`<br>2. Exécutez la requête et vérifiez que les données sont retournées à partir de la vue. |

## 3. Exercice 2 : Création de vues indexées

| Tâche | Informations |
|---|---|
| Créer un nouveau fichier de requête. | 1. Ajoutez une nouvelle requête au projet, en vous connectant au serveur **MIAMI** à l'aide de l'authentification Windows lorsque vous y êtes invité.<br>2. Remplacez le nom du fichier de requête par **CreateViewIndex.sql**. |
| Créer l'index **IX_vEmployeeDetails**. | 1. Dans la fenêtre de la requête, tapez l'instruction Transact-SQL appropriée pour créer l'index cluster unique nommé **IX_vEmployeeDetails** sur la vue **HumanResources.vEmployeeDetails** dans la base de données **AdventureWorks**.<br>2. Exécutez la requête, puis enregistrez le fichier de requête.<br>3. Utilisez l'Explorateur d'objets pour vérifier que l'index a été créé. |

## 4. Exercice 3 : Création de vues partitionnées

| Tâche | Informations |
|---|---|
| Préparer les instances de SQL Server. | 1. Dans l'Explorateur Windows, affichez le contenu du dossier **D:\Labfiles\Starter**.<br>2. Double-cliquez sur **LabSetup.cmd** pour exécuter le script qui configure les serveurs pour la vue partitionnée. |
| Créer un nouveau fichier de requête. | 1. Ajoutez une nouvelle requête au projet, en vous connectant au serveur MIAMI à l'aide de l'authentification Windows lorsque vous y êtes invité.<br>2. Remplacez le nom du fichier de requête par **CreatePartitionedView.sql**. |
| Créer la vue partitionnée de données distribuées **Person.vContact**. | 1. Dans la fenêtre de la requête, tapez l'instruction Transact-SQL appropriée pour créer la vue partitionnée de données distribuées **Person.vContact** dans la base de données **AW_Contacts**.<br>2. Exécutez la requête, puis enregistrez le fichier de requête.<br>3. Utilisez l'Explorateur d'objets pour vérifier que la vue a été créée.<br>4. Créez une nouvelle requête. Lorsque vous y êtes invité, connectez-vous au serveur **MIAMI**.<br>5. Dans la fenêtre de la requête, tapez l'instruction Transact-SQL appropriée pour sélectionner toutes les lignes de la vue **Person.vContact** dans la base de données **AW_Contacts**.<br>6. Exécutez la requête et vérifiez que les résultats sont retournés à partir de la vue partitionnée. |

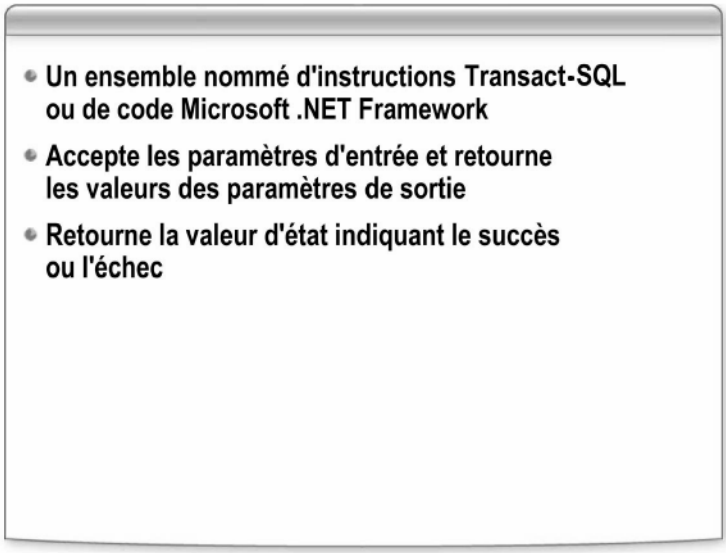
Utilisez la liste de contrôle des résultats ci-après pour vérifier que vous avez correctement effectué cet atelier pratique:

- Création d'une vue nommée **HumanResources.vEmployeeDetails**
- Création d'un index sur la vue **HumanResources.vEmployeeDetails**
- Création d'une vue partitionnée de données distribuées nommée **Person.vContact**.

# MODULE 6 : IMPLEMENTATION DE PROCEDURES STOCKEES ET FONCTIONS

## Leçon 1 : Implémentation de procédures stockées

### 1. Qu'est-ce qu'une procédure stockée ?




Une procédure stockée est un ensemble nommé d'instructions Transact-SQL stocké sur le serveur au sein même de la base de données. Les procédures stockées constituent un moyen d'encapsuler les tâches répétitives ; elles prennent en charge les variables déclarées par l'utilisateur, l'exécution conditionnelle et autres puissantes fonctionnalités de programmation.

Les procédures stockées de Microsoft SQL Server sont similaires aux procédures des autres langages de programmation, en ce sens qu'elles peuvent:

- contenir des instructions qui exécutent des opérations dans la base de données, notamment l'appel d'autres procédures stockées;
- accepter des paramètres d'entrée;
- retourner une valeur d'état à une procédure stockée ou à un lot appelant pour indiquer le succès ou l'échec de l'opération;
- retourner plusieurs valeurs à la procédure stockée appelante ou à l'application cliente sous la forme de paramètres de sortie.

#### 1.1.Avantages

Les procédures stockées offrent de nombreux avantages par rapport à l'exécution de requêtes Transact-SQL appropriées. Elles peuvent effectuer l'une des actions suivantes:

***Encapsuler les fonctionnalités d'entreprise et créer une logique d'application réutilisable***. Les stratégies ou les règles métier encapsulées dans les procédures stockées peuvent être modifiées au sein d'un seul emplacement. Tous les clients peuvent utiliser les mêmes procédures stockées pour garantir la cohérence de l'accès aux données et de leur modification.

***Protéger les utilisateurs contre une exposition des détails des tables de la base de données***. Si un ensemble de procédures stockées prend en charge la totalité des fonctions d'entreprise que les utilisateurs doivent exécuter, ceux-ci n'ont jamais à accéder directement aux tables.

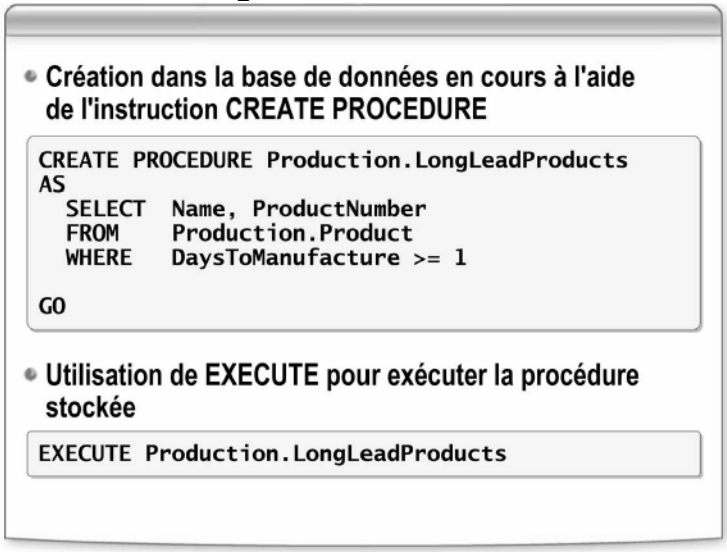
***Fournir des mécanismes de sécurité***. Les utilisateurs doivent être autorisés à exécuter une procédure stockée, même s'ils ne possèdent pas l'autorisation d'accéder aux tables ou aux vues auxquelles la procédure stockée fait référence.

***Améliorer les performances***. Les procédures stockées implémentent nombre de tâches comme un ensemble d'instructions Transact-SQL. La logique conditionnelle peut être appliquée aux résultats des premières instructions Transact-SQL afin de déterminer les prochaines instructions Transact-SQL à exécuter. Toutes les instructions Transact-SQL et la logique conditionnelle deviennent partie intégrante d'un même plan d'exécution sur le serveur.

***Réduire le trafic réseau***. Au lieu d'envoyer des centaines d'instructions Transact-SQL sur le réseau, les utilisateurs peuvent exécuter une opération complexe en adressant une seule instruction, ce qui réduit le nombre de demandes transmises entre le client et le serveur.

***Réduire la vulnérabilité aux attaques de type injection SQL***. L'utilisation de paramètres définis explicitement dans le code SQL supprime le risque qu'un pirate ne soumette des instructions SQL incorporées dans les valeurs des paramètres.

# 2. Syntaxe pour créer des procédures stockées




Vous pouvez créer des procédures stockées à l'aide de l'instruction CREATE PROCEDURE. Les procédures stockées ne peuvent être créées que dans la base de données en cours, à l'exception des procédures stockées temporaires qui le sont toujours dans la base de données **tempdb**. La création d'une procédure stockée est semblable à celle d'une vue. En premier lieu, écrivez et testez les instructions Transact-SQL que vous souhaitez inclure dans la procédure stockée. Puis, si vous obtenez les résultats attendus, créez la procédure stockée.

## 2.1. Syntaxe partielle pour créer une procédure stockée

L'instruction CREATE PROCEDURE contient de nombreuses options, comme illustré dans l'extrait ci-après.

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ [ OUT [ PUT ] ]
     [ ,...n ]
[ WITH <procedure_option> [ ,...n ]
AS sql_statement [;][ ...n ]

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE_AS_Clause ]
```

## 2.2. Exemple de création d'une procédure stockée

L'exemple suivant montre comment vous pouvez créer une procédure stockée simple qui retourne l'ensemble des lignes de tous les produits nécessitant plusieurs jours de fabrication.

```
CREATE PROC Production.LongLeadProducts
AS
  SELECT    Name, ProductNumber
  FROM      Production.Product
  WHERE     DaysToManufacture >= 1
GO
```

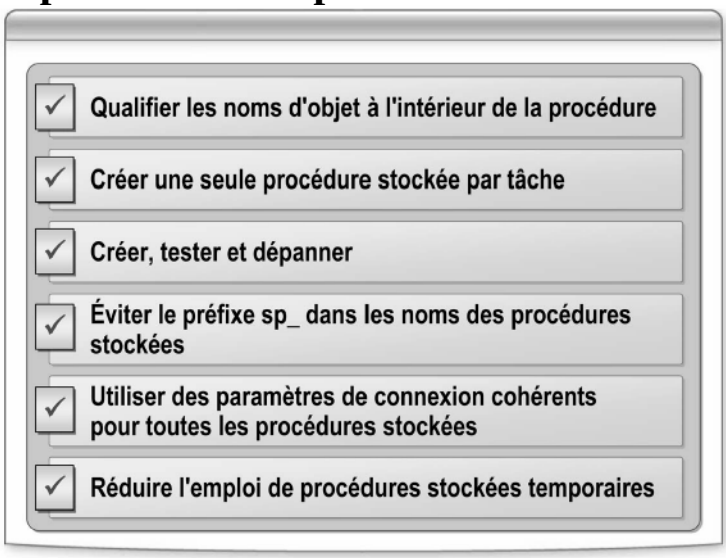
L'exemple précédent crée une procédure intitulée **LongLeadProducts** au sein du schéma **Production**. La commande GO est incluse pour accentuer le fait que les instructions CREATE PROCEDURE doivent être déclarées au sein d'un même lot.

## 2.3. Exemple d'appel de procédure stockée

L'exemple suivant montre comment appeler la procédure stockée **LongLeadProducts**.

```
EXEC Production.LongLeadProducts
```

# 3. Instructions pour créer des procédures stockées




## 3.1. Instructions relatives aux procédures stockées

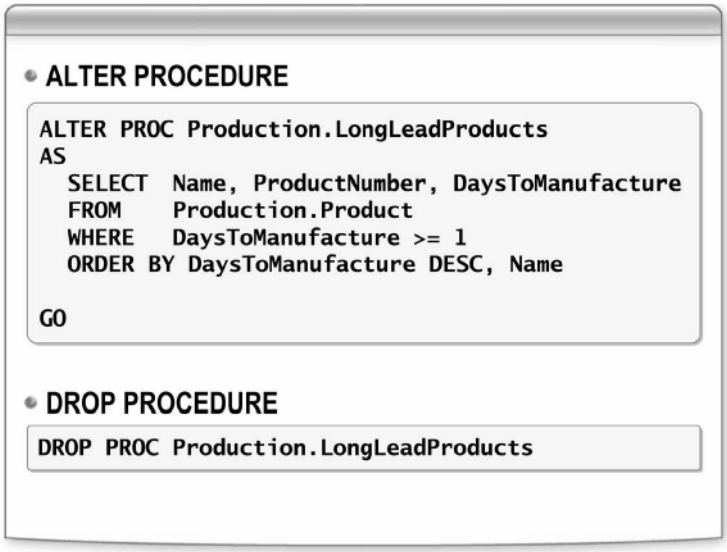
Tenez compte des instructions suivantes lorsque vous créez une procédure stockée:

- Qualifiez les noms d'objets référencés par une procédure stockée avec le nom de schéma approprié. Cette qualification garantit que les tables, les vues ou autres objets de différents schémas sont accessibles au sein de la procédure stockée. Si le nom d'objet référencé n'est pas qualifié, c'est le schéma de la procédure stockée qui est recherché par défaut.
- Concevez chaque procédure stockée pour qu'elle n'accomplisse qu'une seule tâche.
- Créez, testez et dépannez (le cas échéant) votre procédure stockée sur le serveur, puis testez-la à partir de l'ordinateur client.
- Évitez d'utiliser le préfixe **sp_** lorsque vous nommez les procédures stockées locales afin de les distinguer aisément des procédures stockées système. Le fait de ne pas utiliser le préfixe **sp_** pour les procédures stockées d'une base de données locale vous évite des recherches inutiles dans la base de données master. Lorsqu'une procédure stockée dont le nom commence par **sp_** est appelée, SQL Server explore la base de données master avant la base de données locale.

- Utilisez les mêmes paramètres de connexion pour l'ensemble des procédures stockées. SQL Server enregistre les paramètres des deux options SET QUOTED_IDENTIFIER et SET ANSI_NULLS lors de la création ou de la modification d'une procédure stockée. Ces paramètres d'origine sont utilisés lors de l'exécution de la procédure stockée. Par conséquent, tous les paramètres d'une session cliente pour ces options SET sont ignorés lors de l'exécution de la procédure stockée.

N'abusez pas de l'utilisation des procédures stockées temporaires afin d'éviter les conflits sur les tables système de **tempdb**, situation qui peut engendrer une dégradation des performances.

# 4. Syntaxe pour modifier et supprimer des procédures stockées




Les procédures stockées sont souvent modifiées suite aux demandes des utilisateurs ou aux modifications des définitions de table sous-jacentes. Pour modifier une procédure stockée existante et conserver l'affectation des autorisations, utilisez l'instruction ALTER PROCEDURE. SQL Server remplace la définition antérieure de la procédure stockée lorsqu'elle est modifiée en utilisant ALTER PROCEDURE.

Prenez en compte les points suivants lorsque vous utilisez l'instruction ALTER PROCEDURE :

- Si vous souhaitez modifier une procédure stockée créée à l'aide d'une option, comme l'option WITH ENCRYPTION, vous devez inclure l'option dans l'instruction ALTER PROCEDURE pour conserver les fonctionnalités fournies par l'option.
- ALTER PROCEDURE ne modifie qu'une seule procédure. Si votre procédure appelle d'autres procédures stockées, les procédures stockées imbriquées ne sont pas affectées.

## 4.1. Exemple de modification d'une procédure stockée

L'exemple suivant modifie la procédure stockée **LongLeadProducts** afin de sélectionner une colonne supplémentaire et de trier l'ensemble des résultats avec une clause ORDER BY.

```
ALTER PROC Production.LongLeadProducts
AS
    SELECT    Name, ProductNumber, DaysToManufacture
    FROM      Production.Product
    WHERE     DaysToManufacture >= 1
    ORDER BY  DaysToManufacture DESC, Name
GO
```

## 4.2. Suppression d'une procédure stockée

Utilisez l'instruction DROP PROCEDURE pour supprimer de la base de données en cours des procédures stockées définies par l'utilisateur.

Avant de supprimer une procédure stockée, exécutez la procédure stockée **sp_depends** pour déterminer si les objets dépendent ou non de la procédure stockée, comme illustré dans l'exemple ci-après.

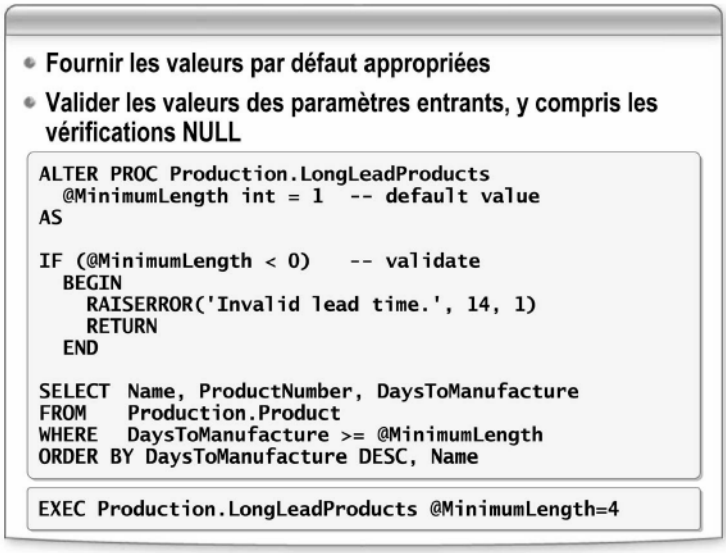
```
EXEC sp_depends @objname = N'Production.LongLeadProducts'
```

L'exemple suivant supprime la procédure stockée **LongLeadProducts**.

```
DROP PROC Production.LongLeadProducts
```

# Leçon 2 : Création de procédures stockées paramétrables

## 1. Paramètres d'entrée




Une procédure stockée communique avec le programme qui appelle la procédure via une liste de 2 100 paramètres au plus. Les paramètres d'entrée permettent de transmettre des valeurs à une procédure stockée, lesquelles peuvent ensuite être utilisées comme variables locales dans la procédure.

### 1.1. Instructions pour utiliser les paramètres d'entrée

Pour définir une procédure stockée qui accepte des paramètres d'entrée, vous déclarez une ou plusieurs variables comme paramètres dans l'instruction CREATE PROCEDURE. Tenez compte des instructions suivantes lors de l'utilisation des paramètres d'entrée:

- Attribuez des valeurs par défaut aux paramètres, si possible. Si vous avez défini une valeur par défaut, un utilisateur peut exécuter la procédure stockée sans spécifier de valeur pour ce paramètre.
- Validez au début de la procédure stockée toutes les valeurs des paramètres entrants pour intercepter au plus tôt les valeurs manquantes ou non valides. Cette opération peut inclure le contrôle de l'éventuelle valeur NULL du paramètre.

### 1.2. Exemple d'utilisation des paramètres d'entrée

L'exemple suivant ajoute un paramètre **@MinimumLength** à la procédure stockée **LongLeadProducts**. De cette façon, la clause WHERE gagne en flexibilité en autorisant l'application appelante à définir le délai approprié.

```
ALTER PROC Production.LongLeadProducts
  @MinimumLength int = 1      -- valeur par défaut
AS

IF (@MinimumLength < 0)      -- valider
   BEGIN
     RAISERROR('Délai non valide.', 14, 1)
     RETURN
END

SELECT Name, ProductNumber, DaysToManufacture
FROM     Production.Product
WHERE     DaysToManufacture >= @MinimumLength
ORDER BY DaysToManufacture DESC, Name
```

La procédure stockée définit une valeur de paramètre par défaut égale à **1** afin que les applications appelantes puissent exécuter la procédure sans spécifier d'argument. Si une valeur est transmise à **@ MinimumLength**, elle est validée pour garantir qu'elle convient à l'objectif de l'instruction SELECT. Si la valeur est inférieure à zéro, une erreur est déclenchée et la procédure stockée retourne immédiatement sans exécuter l'instruction SELECT.

## 1.3. Appels des procédures stockées paramétrables

Vous pouvez définir la valeur d'un paramètre en la passant à la procédure stockée sous forme de position ou de nom de paramètre. Vous ne devez pas mélanger les différents formats lorsque vous fournissez des valeurs.

La spécification d'un paramètre d'une instruction EXECUTE au format *@parameter* = *valeur* est appelée ***passage par nom de paramètre***. Lorsque vous passez les valeurs par nom de paramètre, celles-ci peuvent être spécifiées dans n'importe quel ordre et vous pouvez omettre les paramètres qui autorisent des valeurs null ou ont une valeur par défaut.

L'exemple suivant appelle la procédure stockée **LongLeadProducts** et spécifie le nom de paramètre.

```
EXEC Production.LongLeadProducts @MinimumLength=4
```

Lorsque seules les valeurs sont passées (sans référence aux noms des paramètres auxquels elles sont passées), cette opération est connue sous le nom de *passage des valeurs par position*. Lorsque vous ne spécifiez que les valeurs, les valeurs des paramètres doivent apparaître dans le même ordre que celui dans lequel elles sont définies dans l'instruction CREATE PROCEDURE. Lorsque vous passez les valeurs par position, vous pouvez omettre les paramètres pour lesquels il existe des valeurs par défaut, mais vous ne pouvez pas interrompre la séquence. Par exemple, si une procédure stockée comporte cinq paramètres, vous pouvez omettre les deux derniers paramètres, mais vous ne pouvez pas spécifier le cinquième si vous avez omis le quatrième.

L'exemple suivant appelle la procédure stockée **LongLeadProducts** et spécifie le paramètre en utilisant seulement la position.

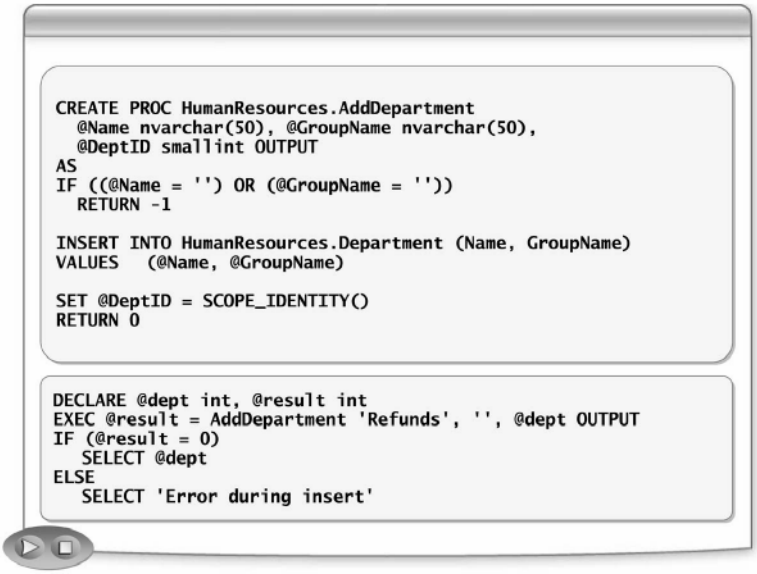
```
EXEC Production.LongLeadProducts 4
```

## 1.4. Utilisation des valeurs par défaut d'un paramètre

La valeur par défaut d'un paramètre, si elle a été définie dans la procédure stockée, est utilisée dans les cas suivants:

- Aucune valeur n'est spécifiée pour le paramètre au moment de l'exécution de la procédure stockée.
- Le mot clé DEFAULT est spécifié comme valeur du paramètre.

# 2. Paramètres de sortie et valeurs de retour




Les procédures stockées peuvent retourner des informations à la procédure stockée appelante ou au client en utilisant à la fois les paramètres de sortie et la valeur de retour.

## 2.1. Caractéristiques des paramètres de sortie

Les paramètres de sortie permettent de conserver les modifications du paramètre qui résultent de l'exécution de la procédure stockée, même après que la procédure stockée a terminé son exécution.

Pour utiliser un paramètre de sortie au sein de Transact-SQL, vous devez spécifier le mot clé OUTPUT dans les deux instructions CREATE PROCEDURE et EXECUTE.

Si le mot clé OUTPUT est omis lorsque la procédure stockée est exécutée, la procédure stockée continue à s'exécuter, mais ne retourne pas la valeur modifiée. Dans la plupart des langages de programmation clients, comme Microsoft Visual C#, le paramètre étant par défaut un paramètre d'entrée, vous devez spécifier la direction du paramètre dans le client.

## 2.2. Exemple d'utilisation des paramètres de sortie

L'exemple suivant crée une procédure stockée qui insère un nouveau service dans la table **HumanResources.Department** de la base de données **AdventureWorks**.

```
CREATE PROC HumanResources.AddDepartment
  @Name nvarchar(50), @GroupName nvarchar(50),
  @DeptID smallint OUTPUT
AS

INSERT INTO HumanResources.Department (Name, GroupName)
VALUES (@Name, @GroupName)

SET @DeptID = SCOPE_IDENTITY()
```

Le paramètre de sortie **@DeptID** stocke l'identité du nouvel enregistrement en appelant la fonction SCOPE_IDENTITY afin qu'une application appelante puisse accéder immédiatement au numéro d'ID généré automatiquement.

L'exemple suivant montre comment l'application appelante peut stocker les résultats de l'exécution de procédure stockée à l'aide de la variable locale **@dept**.

```
DECLARE @dept int
EXEC AddDepartment 'Refunds', '', @dept OUTPUT
SELECT @dept
```

## 2.3. Valeurs de retour

Il est également possible de retourner les informations d'une procédure stockée au moyen de l'instruction RETURN. Cette méthode est plus limitative que l'utilisation de paramètres de sortie, car elle ne retourne qu'une seule valeur entière. L'instruction RETURN est utilisée le plus souvent pour retourner un résultat d'état ou un code d'erreur à partir d'une procédure.

L'exemple suivant modifie la procédure stockée **AddDepartment** pour retourner une valeur spécifiant un succès ou un échec.

```
ALTER PROC HumanResources.AddDepartment
  @Name nvarchar(50), @GroupName nvarchar(50),
  @DeptID smallint OUTPUT
AS
IF ((@Name = '') OR (@GroupName = ''))
  RETURN -1

INSERT INTO HumanResources.Department (Name, GroupName)
VALUES (@Name, @GroupName)

SET @DeptID = SCOPE_IDENTITY()
RETURN 0
```

Si une chaîne vide est transmise à la procédure pour le paramètre **@Name** ou **@GroupName**, la valeur **-1** est retournée pour indiquer un échec. Si l'instruction INSERT réussit, la valeur **0** est retournée pour indiquer un succès.

L'exemple suivant montre comment l'application appelante peut stocker la valeur de retour de l'exécution de procédure stockée à l'aide de la variable locale **@result**.

```
DECLARE @dept int, @result int
EXEC @result = AddDepartment 'Refunds', '', @dept OUTPUT
IF (@result = 0)
    SELECT @dept
ELSE
    SELECT 'Error during insert'
```

**Remarque** SQL Server retourne automatiquement la valeur **0** à partir des procédures stockées si vous ne spécifiez pas votre propre valeur RETURN.

# 3. Application pratique : Création d'une procédure stockée paramétrable

## 3.1. Création d'une procédure stockée simple

Exécutez les étapes suivantes pour créer une procédure stockée simple:

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, puis sur **Microsoft SQL Server 2005**, et cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Dans le menu **Fichier**, pointez sur **Ouvrir**, puis cliquez sur **Fichier**.
4. Ouvrez le fichier de requête **StoredProcedures.sql** enregistré dans le dossier C:\Practices. Connectez-vous au serveur **MIAMI** en utilisant l'authentification Windows lorsque vous y êtes invité.
5. Examinez le code situé sous le commentaire **Vérifier tous les produits**, puis sélectionnez le code et cliquez sur **Exécuter**.
6. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
7. Sélectionnez la requête située sous le commentaire **Tester la procédure stockée**, puis dans la barre d'outils, cliquez sur **Exécuter**.
8. Vérifiez les résultats de la requête.

## 3.2. Création d'une procédure stockée qui accepte un paramètre d'entrée

Exécutez les étapes suivantes pour créer une procédure stockée qui accepte un paramètre d'entrée:

1. Sélectionnez l'instruction ALTER PROCEDURE située sous le commentaire **Altérer la procédure pour obtenir la vérification de produits spécifiques**, puis dans la barre d'outils, cliquez sur **Exécuter**.
2. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
3. Sélectionnez la requête située sous le commentaire **Tester la procédure avec un paramètre**, puis dans la barre d'outils, cliquez sur **Exécuter**.
4. Vérifiez les résultats de la requête. Remarquez que la première exécution a produit les résultats corrects, mais que la seconde exécution a échoué parce que le paramètre n'a pas été passé à la procédure et qu'il ne possède pas de valeur par défaut dans la procédure.
5. Sélectionnez l'instruction ALTER PROCEDURE située sous le commentaire **Altérer la procédure pour obtenir la vérification de produits spécifiques ou de tous les produits**, puis dans la barre d'outils, cliquez sur **Exécuter**.
6. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
7. Sélectionnez la requête située sous le commentaire **Tester la procédure avec un paramètre et une valeur par défaut**, puis dans la barre d'outils, cliquez sur **Exécuter**.
8. Vérifiez les résultats de la requête. Remarquez que la première instruction d'exécution génère des résultats pour le produit spécifié et la seconde instruction pour l'ensemble des produits.

## 3.3. Création d'une procédure stockée qui accepte un paramètre de sortie et retourne des valeurs

Exécutez les étapes suivantes pour créer une procédure stockée qui accepte un paramètre de sortie et retourne une valeur spécifiant un succès ou un échec:

1. Sélectionnez l'instruction ALTER PROCEDURE située sous le commentaire **Altérer la procédure pour obtenir le nombre de vérifications et contrôler si le produit existe**, puis dans la barre d'outils, cliquez sur **Exécuter**.
2. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
3. Sélectionnez les requêtes situées entre le commentaire **Tester les valeurs de sortie et de retour** et l'instruction GO, et dans la barre d'outils, cliquez sur **Exécuter**.
4. Vérifiez les résultats de la requête. Remarquez que les résultats s'affichent correctement, suivis du nombre de vérifications du paramètre de sortie.

5. Modifiez l'instruction EXECUTE en supprimant le mot clé OUTPUT.
6. Sélectionnez à nouveau les requêtes situées entre le commentaire **Tester les valeurs de sortie et de retour** et l'instruction GO, et dans la barre d'outils, cliquez sur **Exécuter**.
7. Vérifiez les résultats de la requête. Remarquez que les résultats s'affichent correctement, mais que le nombre de vérifications retourne la valeur NULL parce que le mot clé OUTPUT a été supprimé.
8. Modifiez l'instruction EXECUTE en remettant le mot clé OUTPUT et en modifiant le paramètre d'entrée de **937** en DEFAULT.
9. Sélectionnez à nouveau les requêtes situées entre le commentaire **Tester les valeurs de sortie et de retour** et l'instruction GO, et dans la barre d'outils, cliquez sur **Exécuter**.
10. Vérifiez les résultats de la requête. Remarquez que toutes les vérifications sont maintenant retournées et que le nombre de vérifications a augmenté.
11. Modifiez l'instruction EXECUTE en remplaçant la valeur DEFAULT du paramètre d'entrée par la valeur **100**.
12. Sélectionnez à nouveau les requêtes situées entre le commentaire **Tester les valeurs de sortie et de retour** et l'instruction GO, et dans la barre d'outils, cliquez sur **Exécuter**.
13. Vérifiez les résultats de la requête. Remarquez qu'aucune vérification n'est retournée et que le message **ProductID does not exist** s'affiche parce qu'un ID de produit non valide a été utilisé.

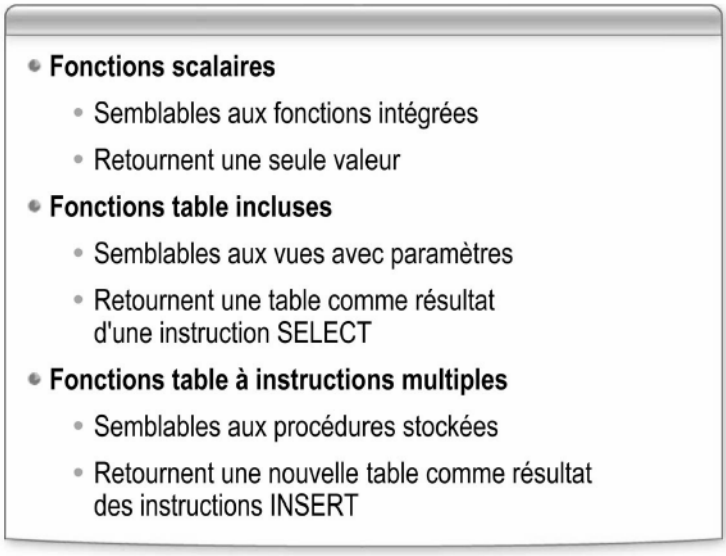
### 3.4. Supprimer une procédure stockée

Exécutez les étapes suivantes pour supprimer une procédure stockée:

1. Sélectionnez l'instruction DROP PROCEDURE située sous le commentaire **Supprimer la procédure**, puis dans la barre d'outils, cliquez sur **Exécuter**.
2. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.

# Leçon 3 : Création de fonction

## 1. Types de fonctions ?




Une fonction est une routine composée d'une ou de plusieurs instructions Transact-SQL qui permettent d'encapsuler le code en vue de sa réutilisation. Une fonction accepte zéro, un ou plusieurs paramètres d'entrée et retourne une *valeur scalaire* ou une *table*. Les paramètres d'entrée peuvent avoir tout type de données, à

l'exception de **timestamp**, **cursor** et **table**, mais les fonctions ne prennent pas en charge les paramètres de sortie.

### Fonctions scalaires

Les fonctions scalaires retournent une valeur de donnée unique dont le type est défini dans la clause RETURNS. La syntaxe de ces types de fonctions est très similaire à celle des fonctions système intégrées telles que COUNT ou MAX.

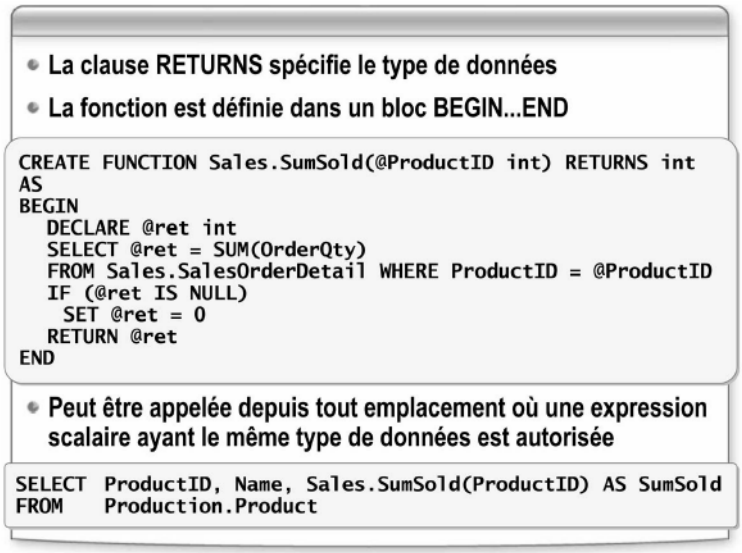
### Fonctions tables incluses

Une fonction table incluse retourne une table qui est le résultat d'une simple instruction SELECT. Une fonction est similaire à une vue, mais offre une plus grande souplesse, car vous pouvez fournir des paramètres à la fonction.

### Fonctions tables à instructions multiples

Une fonction table à instructions multiples retourne une table générée par une ou plusieurs instructions Transact-SQL, et s'apparente à une procédure stockée. Contrairement à une procédure stockée, il peut être fait référence à une fonction table à instructions multiples dans la clause FROM d'une instruction SELECT comme s'il s'agissait d'une vue ou d'une table.

## 2. Qu'est-ce qu'une fonction scalaire ?




Une fonction scalaire retourne une valeur de donnée unique dont le type est défini dans la clause RETURNS. Le corps de la fonction, défini dans le bloc BEGIN...END, contient l'ensemble des instructions Transact-SQL qui retournent la valeur.

L'exemple suivant crée une fonction scalaire qui additionne toutes les ventes d'un produit de la base de données **AdventureWorks** et retourne le total sous forme d'un **int** (entier).

```
CREATE FUNCTION Sales.SumSold(@ProductID int) RETURNS int
AS
BEGIN
    DECLARE @ret int
    SELECT @ret = SUM(OrderQty)
    FROM Sales.SalesOrderDetail WHERE ProductID = @ProductID
    IF (@ret IS NULL)
        SET @ret = 0
    RETURN @ret
END
```

**Remarque** Lors de la modification ou de la suppression d'une fonction, vous utilisez une syntaxe similaire à celle employée pour modifier ou supprimer tout autre objet de base de données. Utilisez ALTER FUNCTION pour modifier vos fonctions après les avoir créées et DROP FUNCTION pour les supprimer.

## 2.1. Appel des fonctions scalaires

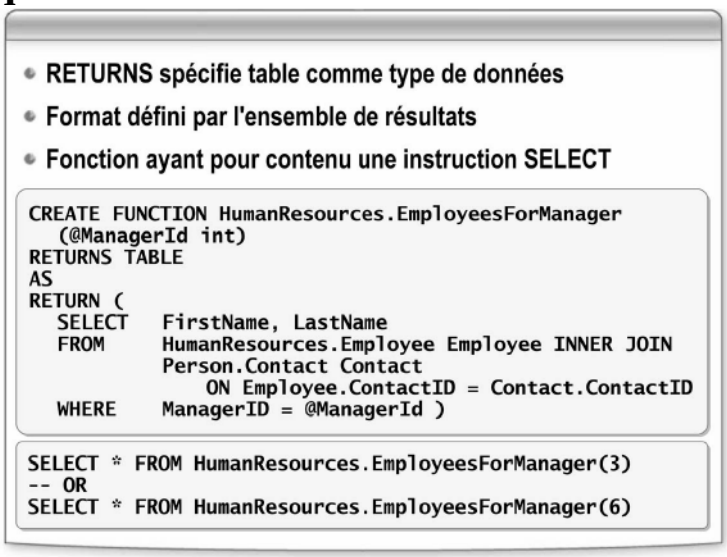
Vous pouvez appeler une fonction définie par l'utilisateur qui retourne une valeur scalaire partout où une expression scalaire ayant le même type de données est autorisée dans les instructions Transact-SQL: Le tableau suivant propose des exemples d'utilisation des fonctions scalaires.

| Domaine | Exemple |
|---|---|
| Requêtes | ■ En tant qu'*expression* dans la liste *select_list* d'une instruction SELECT.<br>■ En tant qu'*expression* ou *string_expression* d'une clause WHERE ou HAVING.<br>■ En tant que *group_by_expression* d'une clause GROUP BY.<br>■ En tant qu'*order_by_expression* d'une clause ORDER BY.<br>■ En tant qu'*expression* de la clause SET d'une instruction UPDATE.<br>■ En tant qu'*expression* de la clause VALUES d'une instruction INSERT. |
| Définition de table | ■ Contraintes CHECK. Les fonctions peuvent uniquement faire référence aux colonnes d'une même table.<br>■ Définitions DEFAULT Les fonctions ne peuvent contenir que des constantes.<br>■ Colonnes calculées. Les fonctions peuvent uniquement faire référence aux colonnes d'une même table. |
| Instructions Transact-SQL | ■ Dans les opérateurs d'affectation<br>■ Dans les expressions booléennes des instructions de contrôle de flux.<br>■ Dans les expressions CASE.<br>■ Dans les instructions PRINT (uniquement pour les fonctions qui retournent une chaîne de caractères). |
| Fonctions et procédures stockées | ■ Comme arguments d'une fonction.<br>■ Comme instruction RETURN d'une procédure stockée (uniquement pour les fonctions scalaires qui retournent un entier).<br>■ Comme instruction RETURN d'une fonction définie par l'utilisateur, à condition que la valeur retournée par la fonction appelée définie par l'utilisateur puisse être convertie implicitement dans le type de données retourné de la fonction appelante. |

L'exemple suivant exécute une instruction SELECT qui extrait les champs **ProductID** et **Name**, ainsi que le résultat de la fonction scalaire **SumSold** pour chaque enregistrement produit de la base de données **AdventureWorks**.

```
SELECT ProductID, Name, Sales.SumSold(ProductID) AS SumSold
FROM     Production.Product
```

# 3. Qu'est-ce qu'une fonction table incluse ?




## 3.1. Quand utiliser les fonctions incluses

Vous pouvez utiliser les fonctions incluses pour offrir une fonctionnalité équivalente à celle des vues paramétrables. L'une des limites des vues est que vous n'êtes pas autorisé à y inclure un paramètre fourni par l'utilisateur lorsque vous les créez. Vous pouvez généralement résoudre ce problème en fournissant une clause WHERE lors de l'appel de la vue. Toutefois, cette action peut nécessiter la création d'une chaîne pour l'exécution dynamique, ce qui peut accroître la complexité de l'application. Vous pouvez offrir les fonctionnalités d'une vue paramétrable en utilisant une fonction table incluse.

Les caractéristiques des fonctions table incluses définies par l'utilisateur sont les suivantes:

- L'instruction RETURNS spécifie **table** comme type de données retourné.
- L'ensemble de résultats de l'instruction SELECT définit le format de la variable de retour.
- La clause RETURN contient une seule instruction SELECT entre parenthèses. L'instruction SELECT utilisée dans une fonction incluse est soumise aux mêmes restrictions que les instructions SELECT utilisées dans les vues.
- Le corps de la fonction ne doit pas se trouver dans un bloc BEGIN...END.

## 3.2. Exemple d'utilisation d'une fonction table incluse

L'exemple suivant crée une fonction table incluse qui retourne les noms des employés d'un chef de service particulier enregistré dans la base de données **AdventureWorks**.

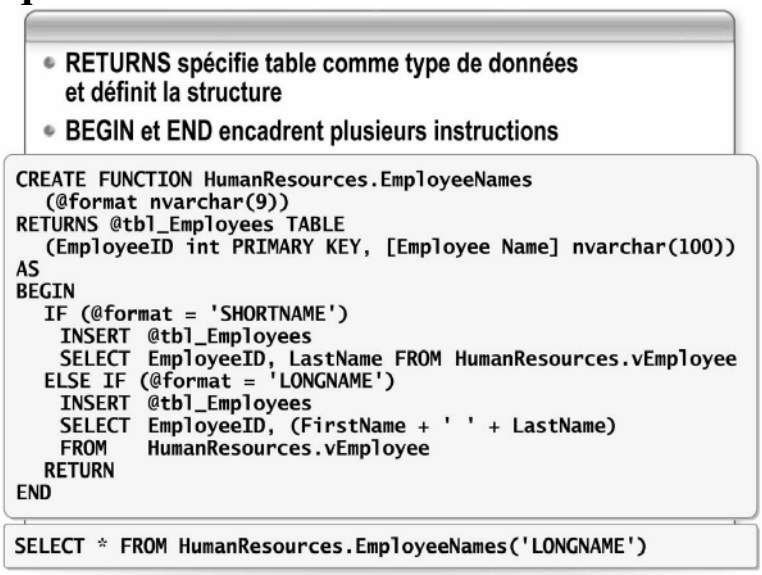
```
CREATE FUNCTION HumanResources.EmployeesForManager
    (@ManagerId int)
RETURNS TABLE
AS
RETURN (
    SELECT    FirstName, LastName
    FROM      HumanResources.Employee Employee INNER JOIN
              Person.Contact Contact
                   ON Employee.ContactID = Contact.ContactID
    WHERE     ManagerID = @ManagerId )
```

### 3.3. Appel des fonctions table incluses

Utilisez une fonction table incluse chaque fois que, normalement, vous utiliseriez une vue, comme dans la clause FROM d'une instruction SELECT. Les exemples suivants extraient tous les noms d'employés de deux chefs de service.

```
SELECT * FROM HumanResources.EmployeesForManager(3)
-- OR
SELECT * FROM HumanResources.EmployeesForManager(6)
```

## 4. Qu'est-ce qu'une fonction table à instructions multiples ?




### 4.1. Quand utiliser les fonctions table à instructions multiples

Une fonction table à instructions multiples constitue la combinaison d'une vue et d'une procédure stockée. Vous pouvez utiliser les fonctions définies par l'utilisateur qui retournent une table pour remplacer les procédures stockées ou les vues.

Une fonction table (comme une procédure stockée) peut utiliser une logique complexe et plusieurs instructions Transact-SQL pour créer une table. De la même façon que vous utilisez une vue, vous pouvez recourir à une fonction table dans la clause FROM d'une instruction Transact-SQL.

Les caractéristiques des fonctions table à instructions multiples sont les suivantes:

- L'instruction RETURNS spécifie **table** comme type de données retourné et définit un nom pour la table ainsi que le format.
- Un bloc BEGIN...END délimite le corps de la fonction.

### 4.2. Exemple d'une fonction table à instructions multiples

L'exemple suivant crée une variable de table **@tbl_Employees** composée de deux colonnes. La seconde colonne varie selon la valeur de paramètre **@format** demandée.

```
CREATE FUNCTION HumanResources.EmployeeNames
     (@format nvarchar(9))
RETURNS @tbl_Employees TABLE
     (EmployeeID int PRIMARY KEY, [Employee Name] nvarchar(100))
AS
BEGIN
    IF (@format = 'SHORTNAME')
        INSERT @tbl_Employees
        SELECT EmployeeID, LastName
        FROM     HumanResources.vEmployee
    ELSE IF (@format = 'LONGNAME')
        INSERT @tbl_Employees
        SELECT EmployeeID, (FirstName + ' ' + LastName)
        FROM     HumanResources.vEmployee
    RETURN
END
```

### 4.3. Appel de fonction table à instructions multiples

Vous pouvez appeler la fonction dans la clause FROM au lieu d'utiliser une table ou une vue. Les exemples suivants extraient les noms des employés dans les formats long et court.

```
SELECT * FROM HumanResources.EmployeeNames('LONGNAME')
-- OR
SELECT * FROM HumanResources.EmployeeNames('SHORTNAME')
```

# 5. Application pratique : Création de fonctions

## 5.1. Création d'une fonction scalaire

Exécutez les étapes suivantes pour créer une fonction scalaire:

1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, puis sur **Microsoft SQL Server 2005**, et cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Dans le menu **Fichier**, pointez sur **Ouvrir**, puis cliquez sur **Fichier**.
4. Ouvrez le fichier de requête **UserDefinedFunctions.sql** enregistré dans le dossier C:\Practices.
5. Sélectionnez le code situé sous le commentaire **Créer une fonction scalaire**, puis dans la barre d'outils, cliquez sur **Exécuter**.
6. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
7. Sélectionnez la requête situé sous le commentaire **Tester la fonction scalaire dans SELECT**, puis dans la barre d'outils, cliquez sur **Exécuter**.
8. Vérifiez les résultats de la requête. Notez que la plupart des produits n'ont aucune vérification et affichent, par conséquent, une évaluation moyenne égale à **0**.
9. Sélectionnez la requête situé sous le commentaire **Tester la fonction scalaire dans WHERE**, puis dans la barre d'outils, cliquez sur **Exécuter**.
10. Vérifiez les résultats de la requête. Remarquez que seuls sont retournés les produits avec des vérifications.

## 5.2. Création d'une fonction table incluse

Exécutez les étapes suivantes pour créer une fonction table incluse:

1. Sélectionnez l'instruction CREATE FUNCTION située sous le commentaire **Créer une fonction incluse**, puis dans la barre d'outils, cliquez sur **Exécuter**.
2. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
3. Sélectionnez la requête située sous le commentaire **Tester la fonction incluse**, puis dans la barre d'outils, cliquez sur **Exécuter**.
4. Vérifiez les résultats de la requête. Remarquez que seules les vérifications d'un produit sont affichées.

### 5.3. Création d'une fonction table à instructions multiples

Exécutez les étapes suivantes pour créer une fonction table à instructions multiples:

1. Sélectionnez l'instruction CREATE FUNCTION située sous le commentaire **Créer une fonction à plusieurs instructions**, puis dans la barre d'outils, cliquez sur **Exécuter**.
2. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
3. Sélectionnez les requêtes situées entre le commentaire **Tester la fonction à plusieurs instructions** et l'instruction GO, puis dans la barre d'outils, cliquez sur **Exécuter**.
4. Vérifiez les résultats de la requête. Notez que le premier ensemble de résultats contient les évaluations de trois produits ou plus et le second les résultats de deux produits ou moins.

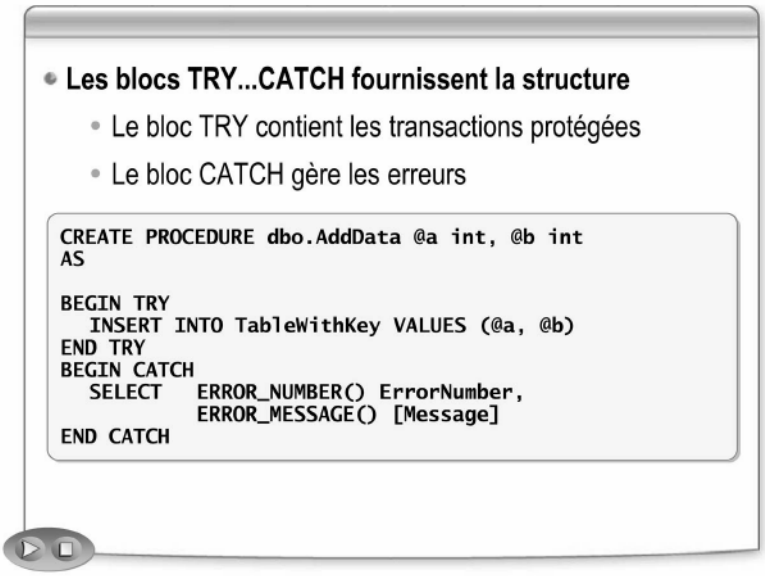
### 5.4. Création d'une fonction table à instructions multiples

Exécutez les étapes suivantes pour supprimer une fonction définie par l'utilisateur:

1. Sélectionnez l'instruction DROP FUNCTION située sous le commentaire **Supprimer les fonctions définies par l'utilisateur**, puis dans la barre d'outils, cliquez sur **Exécuter**.
2. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
3. Fermez SQL Server Management Studio sans enregistrer les modifications apportées au fichier.

## Leçon 4 : Gestion des erreurs

## 1. Syntaxe pour la gestion structurée des exceptions ?




La gestion structurée des exceptions est une solution couramment adoptée dans nombre de langages de programmation, comme Microsoft Visual Basic® et Visual C#, pour gérer les exceptions. SQL Server 2005

vous permet d'utiliser la gestion structurée des exceptions dans toute situation transactionnelle, telle qu'une procédure stockée. Votre code est ainsi plus lisible et plus facile à maintenir.

### 1.1. Syntaxe

Utilisez les blocs TRY...CATCH pour implémenter la gestion structurée des exceptions. Le bloc TRY contient le code transactionnel susceptible d'échouer. Le bloc CATCH contient le code qui s'exécute si une erreur se produit dans le bloc TRY.

Le bloc TRY...CATCH obéit à la syntaxe suivante:

```
BEGIN TRY
   { sql_statement | statement_block }
END TRY
BEGIN CATCH
   { sql_statement | statement_block }
END CATCH
```

La partie *sql_statement* ou *statement_block* de la syntaxe désigne toute instruction ou tout groupe d'instructions Transact-SQL.

### 1.2. Exemple de bloc TRY … CATCH

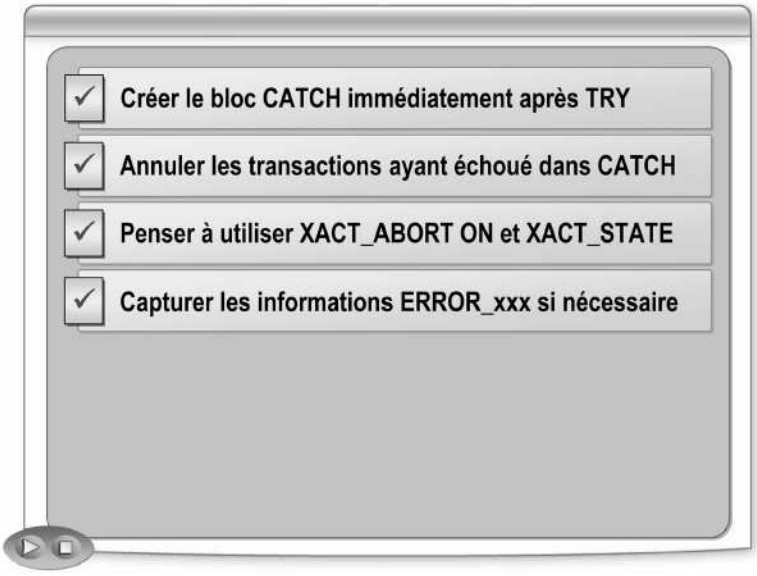
Dans cet exemple, la procédure stockée **AddData** essaie d'insérer deux valeurs dans la table **TestData**. La première colonne de la table **TestData** est une clé primaire de type entier, et la seconde colonne contient des données de type entier. Le bloc TRY. .. CATCH de la procédure stockée **AddData** protège l'instruction INSERT **TestData** et retourne les numéros et message d'erreur comme partie intégrante de la logique du bloc CATCH à l'aide des fonctions ERROR_NUMBER et ERROR_MESSAGE.

```
CREATE TABLE dbo.TableWithKey (ColA int PRIMARY KEY, ColB int)
GO

CREATE PROCEDURE dbo.AddData @a int, @b int
AS
BEGIN TRY
    INSERT INTO TableWithKey VALUES (@a, @b)
END TRY
BEGIN CATCH
     SELECT ERROR_NUMBER() ErrorNumber, ERROR_MESSAGE() [Message]
END CATCH
GO

EXEC dbo.AddData 1, 1
EXEC dbo.AddData 2, 2
EXEC dbo.AddData 1, 3          --viole la clé primaire
```

## 2. Instructions pour gérer les erreurs




## 2.1. Création du bloc CATCH

Créez le bloc CATCH immédiatement après l'instruction END TRY à l'aide des instructions BEGIN CATCH et END CATCH. Vous ne pouvez pas inclure d'autres instructions entre les instructions END TRY et BEGIN CATCH.

L'exemple suivant ne pourra pas être compilé.

```
BEGIN TRY
    -- INSERT INTO ...
END TRY
SELECT * FROM TableWithKey -- NOT ALLOWED
BEGIN CATCH
    -- SELECT ERROR_NUMBER()
END CATCH
```

## 2.2. Echec de l'annulation des transactions

L'utilisation de transactions vous permet de regrouper plusieurs instructions de telle sorte qu'elles se déroulent toutes avec succès ou qu'aucune d'elles ne s'achève avec succès. Considérez l'exemple suivant, qui n'utilise pas de transactions.

```
CREATE TABLE dbo.TableNoKey (ColA int, ColB int)
CREATE TABLE dbo.TableWithKey (ColA int PRIMARY KEY, ColB int)
GO

CREATE PROCEDURE dbo.AddData @a int, @b int
AS
BEGIN TRY
    INSERT dbo.TableNoKey VALUES (@a, @b)
    INSERT dbo.TableWithKey VALUES (@a, @b)
END TRY
BEGIN CATCH
     SELECT ERROR_NUMBER() ErrorNumber, ERROR_MESSAGE() [Message]
END CATCH
GO

EXEC dbo.AddData 1, 1
EXEC dbo.AddData 2, 2
EXEC dbo.AddData 1,  3          --violates the primary key
```

Cet exemple exécute séquentiellement deux insertions dans deux tables différentes. La première insertion réussit toujours parce qu'il n'y a aucune contrainte de clé primaire sur la table. La deuxième insertion échoue

chaque fois qu'une valeur **ColA** dupliquée est insérée. Comme les transactions ne sont pas utilisées dans cet exemple, la première insertion réussit même si la seconde échoue, ce qui peut provoquer des résultats inattendus.

L'exemple suivant utilise les transactions afin de garantir qu'aucune insertion ne réussit si l'une des insertions échoue. À cette fin, l'exemple emploie les instructions BEGIN TRAN et COMMIT TRAN au sein du bloc TRY et une instruction ROLLBACK TRAN au sein du bloc CATCH.

```
ALTER PROCEDURE dbo.AddData @a int, @b int
AS
BEGIN TRY
    BEGIN TRAN
    INSERT dbo.TableNoKey VALUES (@a, @b)
    INSERT dbo.TableWithKey VALUES (@a, @b)
    COMMIT TRAN
END TRY
BEGIN CATCH
    ROLLBACK TRAN
    SELECT ERROR_NUMBER() ErrorNumber, ERROR_MESSAGE() [Message]
END CATCH
GO
```

## 2.3. Utilisation de XACT_ABORT et XACT_STATE

L'option XACT_ABORT indique si SQL Server annule automatiquement la transaction en cours lorsqu'une instruction Transact-SQL déclenche une erreur d'exécution. Toutefois, si l'erreur se produit dans un bloc TRY, la transaction n'est pas automatiquement annulée; elle devient non validable.

Le code d'un bloc CATCH doit tester l'état d'une transaction à l'aide de la fonction XACT_STATE.
XACT_STATE retourne la valeur **–1** si une transaction non validable est présente dans la session en cours.
Le bloc CATCH ne doit pas tenter de valider la transaction et doit la restaurer manuellement.
XACT_STATE retourne une valeur égale à **1** quand une transaction peut être validée en toute sécurité. La valeur **0** signifie qu'il n'y a aucune transaction en cours.

L'exemple suivant définit XACT_ABORT avec la valeur ON et teste l'état de la transaction dans le bloc CATCH.

```
SET XACT_ABORT ON
BEGIN TRY
     BEGIN TRAN
     ...
     COMMIT TRAN
END TRY
BEGIN CATCH
     IF (XACT_STATE()) = -1         -- non validable
         ROLLBACK TRAN
     ELSE IF (XACT_STATE()) = 1        -- validable
         COMMIT TRAN
END CATCH
```

## 2.4. Captures éventuelles des informations sur les erreurs

SQL Server fournit plusieurs fonctions que vous pouvez appeler dans votre bloc CATCH pour enregistrer les informations sur les erreurs. Par exemple, vous pouvez appeler ces méthodes et stocker les résultats dans une table du journal des erreurs.
Le tableau suivant décrit les fonctions relatives aux erreurs.

| Fonction | Description |
|---|---|
| ERROR_LINE | Retourne le numéro de ligne auquel l'erreur qui s'est produite a provoqué l'exécution du bloc de code CATCH. |
| ERROR_MESSAGE | Retourne des informations de diagnostic sur la cause de l'erreur. De nombreux messages d'erreur possèdent des variables de substitution contenant des informations telles que le nom de l'objet à l'origine de l'erreur. |
| ERROR_NUMBER | Retourne le numéro de l'erreur qui s'est produite. |
| ERROR_PROCEDURE | Retourne le nom de la procédure stockée ou du déclencheur dans lequel l'erreur s'est produite. |
| ERROR_SEVERITY | Retourne une valeur indiquant le degré de gravité de l'erreur. Les erreurs dont le degré de gravité est faible (1 ou 2) constituent de simples messages d'information ou des avertissements peu importants. Si le degré de gravité est élevé, le problème doit être résolu le plus rapidement possible. |
| ERROR_STATE | Retourne la valeur d'état. Certains messages d'erreur peuvent apparaître en de multiples endroits du code du moteur de base de données. Un code d'état unique est assigné à chaque condition spécifique qui déclenche une erreur. Ces informations peuvent être utiles lorsque vous travaillez avec les articles de la Base de connaissance Microsoft pour déterminer si le problème enregistré est identique à l'erreur que vous avez rencontrée. |

# 3. Application pratique : gestion des erreurs

## 3.1. Ajout de la gestion des erreurs à une procédure stockée

Exécutez les étapes suivantes pour ajouter la gestion des erreurs à une procédure stockée:

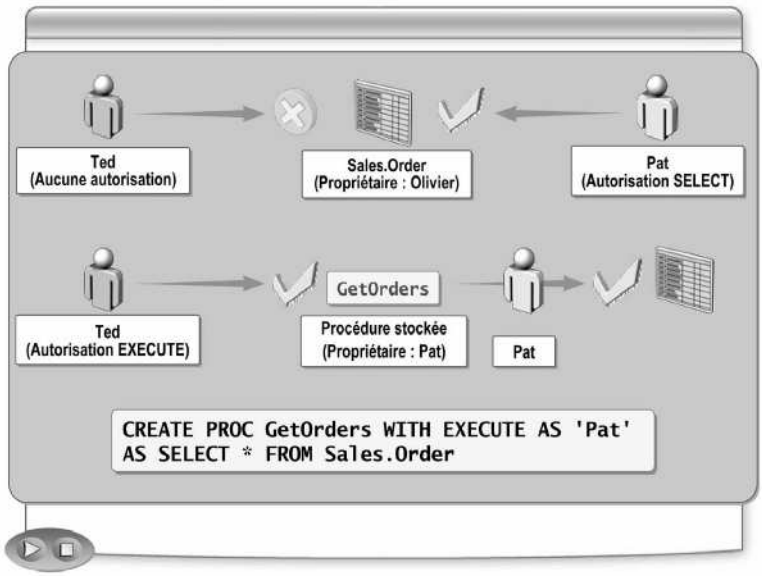
1. Cliquez sur **Démarrer**, pointez sur **Tous les programmes**, puis sur **Microsoft SQL Server 2005**, et cliquez sur **SQL Server Management Studio**.
2. Dans la boîte de dialogue **Se connecter au serveur**, puis cliquez sur **Se connecter**.
3. Dans le menu **Fichier**, pointez sur **Ouvrir**, puis cliquez sur **Fichier**.
4. Ouvrez le fichier de requête **ErrorHandling.sql** enregistré dans le dossier C:\Practices. *
5. Sélectionnez le code situé sous le commentaire **Créer la procédure sans gestion des erreurs**, puis dans la barre d'outils, cliquez sur **Exécuter**.
6. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
7. Sélectionnez les requêtes EXECUTE et SELECT situées sous le commentaire **Tester la procédure stockée sans gestion des erreurs**, puis dans la barre d'outils, cliquez sur **Exécuter**.
8. Vérifiez les résultats de la requête. Notez que la vérification a été ajoutée avec succès à la table.
9. Modifiez la valeur du paramètre **@Rating** de **4** en **10**.
10. Sélectionnez à nouveau les requêtes EXECUTE et SELECT situées sous le commentaire **Tester la procédure stockée sans gestion des erreurs**, puis dans la barre d'outils, cliquez sur **Exécuter**.
11. Vérifiez les résultats de la requête. Remarquez que l'insertion échoue à cause d'une contrainte CHECK sur la

colonne des évaluations.

12. Sélectionnez l'instruction ALTER PROCEDURE située sous le commentaire **Implémenter la gestion des erreurs**, puis dans la barre d'outils, cliquez sur **Exécuter**.
13. Vérifiez la sortie de la requête et confirmez que la commande s'est déroulée avec succès.
14. Sélectionnez la requête EXECUTE située sous le commentaire **Tester la procédure stockée avec gestion des erreurs**, puis dans la barre d'outils, cliquez sur **Exécuter**.
15. Vérifiez les résultats de la requête. Remarquez que le numéro d'erreur et le message s'affichent parce que le bloc CATCH gère l'erreur avec succès.
16. Sélectionnez l'instruction DROP PROCEDURE à la fin du fichier, puis cliquez sur **Exécuter**.

# Leçon 5 : Contrôle du contexte d'exécution

## 1. Qu'est-ce que le contexte d'exécution ?




### 1.1. Définition

Le contexte d'exécution établit l'identité par rapport à laquelle les autorisations sont vérifiées. L'utilisateur ou l'ouverture de session qui appelle un module, tel qu'une procédure stockée ou une fonction, détermine habituellement le contexte d'exécution.

### 1.2. Exemple de contexte d'exécution

Le contexte d'exécution désigne l'identité utilisée par le code lorsqu'il est exécuté ; par défaut, il s'agit de l'appelant, ce qui peut créer des problèmes s'il y a une rupture dans la chaîne de propriétés, comme décrit dans l'exemple suivant.

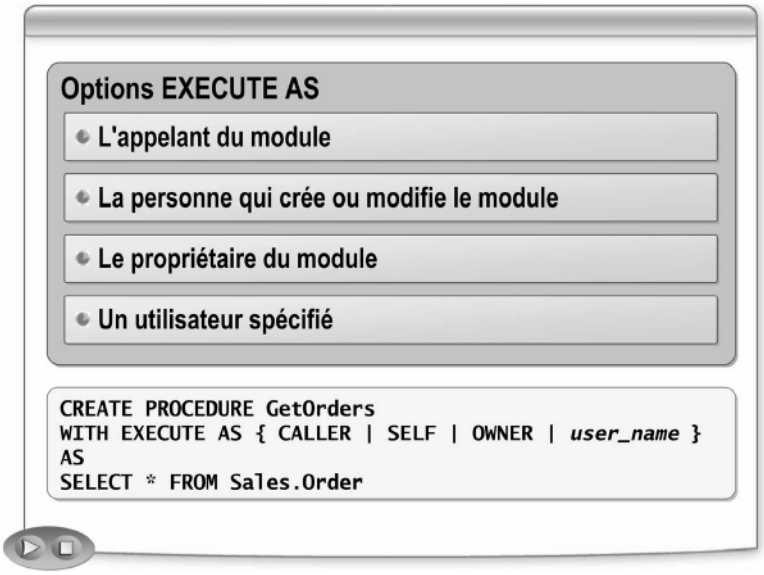
Si l'utilisateur **Olivier** possède une table intitulée **Sales.Order** et qu'il n'accorde les autorisations SELECT qu'à l'utilisateur **Pat**, l'utilisateur **Ted** ne peut pas accéder à la table (voir l'illustration).

Si **Pat** possède une procédure stockée intitulée **GetOrders** qui lit les données de la table **Sales.Order**, **Pat** peut exécuter cette procédure stockée sans aucun problème de sécurité, car **Pat** possède les autorisations SELECT sur la table.

Si **Pat** accorde à **Ted** l'autorisation EXECUTE sur la procédure stockée, il en résulte une erreur lorsque **Ted** essaie d'exécuter la procédure. La raison en est que, par défaut, la procédure stockée s'exécute en tant que **Ted** et que celui-ci ne possède pas l'autorisation SELECT sur la table **Sales.Order**.

Pour permettre à **Ted** d'exécuter la procédure stockée avec succès, le contexte d'exécution doit être modifié avec un utilisateur qui dispose des autorisations appropriées requises. Vous pouvez utiliser la clause EXECUTE AS pour passer d'un contexte d'exécution de la procédure stockée à l'autre, comme expliqué dans la rubrique suivante.

## 2. Clause EXECUTE AS ?




Vous pouvez utiliser la clause EXECUTE AS dans une procédure stockée ou une fonction pour définir l'identité utilisée dans son contexte d'exécution. Une bonne maîtrise de l'utilisation de la clause EXECUTE AS peut vous aider à implémenter la sécurité dans les scénarios où vous devez accéder à des objets dépendants, tout en ne souhaitant pas vous appuyer sur les chaînes de propriétés non rompues.

Vous pouvez utiliser la clause EXECUTE AS avec toute instruction CREATE PROCEDURE ou CREATE FUNCTION, à l'exception des déclarations de fonctions table incluses.

La syntaxe de la clause EXECUTE AS est la suivante:

```
EXECUTE AS { CALLER | SELF | OWNER | user_name }
```

Les options incluses dans la syntaxe de la clause EXECUTE AS sont décrites dans le tableau suivant.

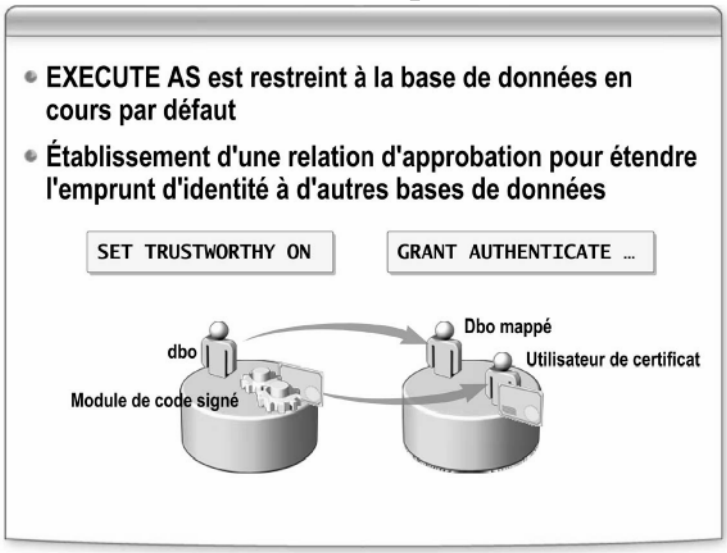
| Option | Description |
|---|---|
| CALLER | Exécution en utilisant l'identité de l'utilisateur appelant. Il s'agit de la valeur par défaut. |
| SELF | Exécution en utilisant l'identité de l'utilisateur qui crée ou modifie la procédure stockée ou la fonction. Cette valeur est non affectée si un autre utilisateur prend possession du module. |
| OWNER | Exécution en utilisant l'identité du propriétaire de la fonction. Cette valeur change si un autre utilisateur prend possession du module. |
| *user_name* | Exécution en utilisant l'identité de l'utilisateur spécifié. Si *user_name* est identique à l'utilisateur qui crée ou modifie le module, cette option est équivalente à EXECUTE AS SELF. |

**Exemple d'utilisation**

Pour résoudre le problème décrit dans la précédente rubrique, la spécification de *Pat* comme **user_name** permet à **Ted** d'exécuter la procédure stockée avec succès, comme illustré dans l'exemple ci-après.

```
CREATE PROCEDURE GetOrders WITH EXECUTE AS 'Pat' AS SELECT * FROM Sales.Order
```

# 3. Options d'extension du contexte d'emprunt d'identité




## 3.1. Limite de EXECUTE AS

Lorsque vous utilisez la clause EXECUTE AS pour modifier le contexte d'exécution afin qu'un module de code s'exécute en tant qu'un autre utilisateur que l'appelant, on dit que le code « emprunte l'identité » de l'autre utilisateur. Par défaut, le contexte d'emprunt d'identité qui en résulte n'est valide qu'au sein de l'étendue de la base de données en cours. Cela signifie que si vous créez une procédure stockée qui appelle une table d'une autre base de données, la clause EXECUTE AS transmet le contexte d'emprunt d'identité à l'autre base de données, mais que le contexte ne sera pas valide.

## 3.2. Etablissement d'une relation d'approbation pour prolonger l'emprunt d'identité

Vous pouvez prolonger sélectivement l'étendue de l'emprunt d'identité de base de données établi au sein d'une base de données en établissant un modèle d'approbation entre les deux bases de données. Cette solution peut être utile dans le cas d'une application qui utilise deux bases de données et a besoin d'accéder à l'une d'elles à partir de l'autre.

SQL Server 2005 utilise les authentificateurs pour déterminer si un contexte établi est valide au sein d'une étendue particulière. Généralement, l'authentificateur est l'administrateur système ou l'instance de SQL Server – ou l'utilisateur **dbo** dans les bases de données. Concrètement, il est le propriétaire de l'étendue dans laquelle le contexte d'un utilisateur ou d'un nom de connexion particulier est établi.

La validité du contexte de l'utilisateur dont l'identité a été empruntée en dehors de l'étendue de la base de données varie selon que l'authentificateur du contexte est approuvé ou non dans l'étendue cible. Cette approbation est établie en créant une connexion authentificateur dupliquée et en accordant à l'authentificateur l'autorisation AUTHENTICATE si l'étendue cible est une autre base de données ou l'autorisation AUTHENTICATE SERVER si l'étendue cible est une instance de SQL Server. La base de données appelante doit être identifiée comme TRUSTWORTHY

La procédure de base pour établir une relation d'approbation pour l'utilisateur **dbo** est la suivante:

1. Créez un utilisateur dans la base de données cible avec la même ouverture de session que l'utilisateur **dbo** de la base de données appelante. Il s'agira de l'authentificateur dans la base de données cible.
2. Accordez à l'utilisateur mappé l'autorisation AUTHENTICATE (ou AUTHENTICATE SERVER) dans la base de données cible.
3. Accordez à l'authentificateur les autorisations requises par la procédure stockée dans la base de

données appelante.

4. Modifiez la base de données appelante en définissant l'option TRUSTWORTHY avec la valeur ON.

### 3.3. Utilisation de certificats et de la signature pour valider les appelants

Une autre façon d'établir une relation d'approbation entre des bases de données consiste à utiliser les certificats ou les clés asymétriques comme authentificateurs. Vous tirez ainsi parti de la technique dite de signature.

La signature d'un module vérifie que le code à l'intérieur d'un module ne peut être modifié que par une personne ayant accès à la clé privée utilisée pour signer le module. Cela vous permet de définir une relation d'approbation avec le certificat utilisé pour
la signature plutôt qu'avec le seul propriétaire de la base de données. L'approbation du module signé s'effectue en accordant l'autorisation AUTHENTICATE ou AUTHENTICATE SERVER à l'utilisateur de l'étendue cible qui est mappée au certificat ou à la clé asymétrique.

La procédure de base pour utiliser les certificats et la signature afin de valider les appelants est la suivante:

1. Créez un certificat en utilisant l'instruction CREATE CERTIFICATE dans la base de données appelante.
2. Ajoutez une signature à la procédure stockée appelante à l'aide d'ADD SIGNATURE.
3. Importez le certificat dans la base de données cible. Vous pouvez, à cette fin, sauvegarder le certificat de la base de données appelante dans un fichier, puis utiliser CREATE CERTIFICAT dans la base de données cible, en faisant référence au fichier de certificat sauvegardé.
4. Créez un utilisateur basé sur le certificat en utilisant la clause FROM CERTIFICATE de l'instruction CREATE USER et accordez les autorisations utilisateur appropriées sur les objets requis de la base de données.
5. Attribuez l'autorisation AUTHENTICATE ou AUTHENTICATE SERVER à l'utilisateur en fonction du certificat.

# Atelier pratique : Implémentation de procédures stockées et de fonctions

## 1. Scénario

Adventure Works gère tout au long de l'année une liste d'offres spéciales et de remises pour divers produits et qui s'applique aux clients et aux revendeurs. Actuellement, ces informations ne sont directement accessibles qu'à partir de la table **Sales.SpecialOffer**. Un nouvel impératif requiert que ces informations soient extraites à l'aide de procédures stockées et de fonctions définies par l'utilisateur, et que l'insertion de nouvelles offres spéciales s'effectue à l'aide d'une procédure stockée.

Le développeur de base de données senior vous a fourni un projet de scripts SQL Server intitulé **AWProgrammability.ssmssln** et stocké dans le dossier C:\Labfiles\Starter ; il a également formulé les exigences suivantes quant aux modifications que vous devez effectuer:

- Créez une procédure stockée nommée **GetDiscounts** dans le schéma **Sales** qui extrait les colonnes suivantes de **Sales.SpecialOffer** : **Description**, **DiscountPct**, **Type**, **Category**, **StartDate**, **EndDate**, **MinQty** et **MaxQty**. La procédure doit retourner toutes les lignes triées par **Startdate** et **Enddate**.
- Créez une procédure stockée nommée **GetDiscountsForCategory** dans le schéma **Sales** qui accepte un paramètre d'entrée intitulé **@Category**, dont le type de données **nvarchar** accepte jusqu'à **50** caractères. La procédure extrait les mêmes colonnes que pour **GetDiscounts**, mais doit filtrer les lignes en fonction du paramètre **@Category**.

- Créez une procédure stockée nommée **GetDiscountsForCategoryAndDate** dans le schéma **Sales** qui accepte le paramètre **@Category**, comme pour la procédure **GetDiscountsForCategory**, mais inclut un paramètre d'entrée **@DateToCheck** de type **datetime**.

Le paramètre **@DateToCheck** doit pouvoir accepter une valeur NULL par défaut. Si une valeur NULL est spécifiée pour le paramètre **@DateToCheck**, définissez la valeur du paramètre avec la date et l'heure actuelles à l'aide de la fonction GETDATE.

La procédure extrait les mêmes colonnes que pour **GetDiscounts**, mais doit filtrer les lignes en fonction des paramètres **@Category** et **@DateToCheck**.

- Créez une procédure stockée nommée **AddDiscount** dans le schéma **Sales** qui insère de nouveaux enregistrements dans la table **Sales.SpecialOffer**. Le tableau suivant répertorie les paramètres requis pour l'insertion:

| Nom du paramètre | Type de données (input sauf spécification contraire) |
|---|---|
| @Description | nvarchar(255) |
| @DiscountPct | smallmoney |
| @Type | nvarchar(50) |
| @Category | nvarchar(50) |
| @StartDate | datetime |
| @EndDate | datetime |
| @MinQty | int |
| @MaxQty | int |
| @NewProductID | int OUTPUT |

L'instruction INSERT doit être protégée par une gestion des erreurs appropriée et toute erreur doit être enregistrée dans la table **dbo.ErrorLog**. Si la nouvelle insertion réussit, le paramètre **@NewProductID** doit être mis à jour avec la valeur de la fonction SCOPE_IDENTITY. Une valeur de retour doit également indiquer si l'insertion a réussi ou échoué.

- Créez une fonction scalaire définie par l'utilisateur nommée **GetMaximumDiscountForCategory** dans le schéma **Sales** qui extrait le pourcentage maximal de remise actuellement disponible pour une catégorie spécifique. Créez un paramètre **@Category nvarchar(50)** pour limiter les résultats en fonction de la catégorie, et utilisez la fonction GETDATE pour limiter les lignes selon que la remise est actuellement disponible ou pas.

- Créez une fonction table incluse définie par l'utilisateur nommée **GetDiscountsForDate** dans le schéma **Sales**, qui extrait les mêmes colonnes que la procédure stockée **GetDiscounts**. La fonction accepte un paramètre **@DateToCheck datetime** pour filtrer les remises en fonction de la date fournie. Adventure Works peut ainsi tester les remises qui seront disponibles à une date spécifique.
- Créez une fonction table à instructions multiples définie par l'utilisateur nommée **GetDiscountedProducts** dans le schéma **Sales**, qui utilise une requête complexe pour extraire les produits bénéficiant d'une remise. Cette requête complexe vous est fournie. La fonction accepte un paramètre **@IncludeHistory bit** pour filtrer la table retournée selon que les informations sur l'historique des remises sont obligatoires ou que seules les informations en cours sont exigées. La table retournée comporte la définition ci-après.

| Nom de la colonne | Type de données |
|---|---|
| ProductID | int |
| Name | nvarchar(50) |
| ListPrice | money |
| DiscountDescription | nvarchar(255) |
| DiscountPercentage | smallmoney |
| DiscountAmount | money |
| DiscountedPrice | money |

## 2. Exercice 1 : Création de procédures stockées

**Création des procédures stockées GetDiscounts, GetDiscountsForCategory, GetDiscountsForCategory AndDate et AddDiscount**

| Tâche | Informations |
|---|---|
| Créer et tester la procédure stockée **GetDiscounts**. | 1. Ouvrez SQL Server Management Studio. Connectez-vous au serveur **MIAMI** lorsque vous y êtes invité.<br>2. Ouvrez la solution **AWProgrammability.ssmssln** enregistrée dans **D:\Labfiles\Starter**.<br>3. Ouvrez le fichier de requête **InitializeData.sql**, puis exécutez-le, en contrôlant qu'aucune erreur ne se produit.<br>4. Ouvrez le fichier de requête **StoredProcedures.sql**.<br>5. Créez la procédure stockée **Sales.GetDiscounts** conformément aux impératifs, puis exécutez votre instruction CREATE PROCEDURE, en contrôlant qu'aucune erreur ne se produit.<br>6. Testez la procédure stockée **Sales.GetDiscounts**, puis vérifiez les résultats. |
| Créer et tester la procédure stockée **GetDiscountsForCategory**. | 1. Créez la procédure stockée **Sales.GetDiscountsForCategory** conformément aux impératifs, puis exécutez votre instruction CREATE PROCEDURE, en contrôlant qu'aucune erreur ne se produit.<br>2. Testez la procédure stockée **Sales.GetDiscountsForCategory** en utilisant **'Reseller'** comme valeur de **@Category**, puis vérifiez les résultats. |
| Créer et tester la procédure stockée **GetDiscountsFor CategoryAndDate**. | 1. Créez la procédure stockée **Sales.GetDiscountsForCategoryAndDate** conformément aux impératifs, puis exécutez votre instruction CREATE PROCEDURE, en contrôlant qu'aucune erreur ne se produit.<br>2. Testez la procédure stockée **Sales.GetDiscountsForCategoryAndDate** en utilisant **'Reseller'** comme valeur de **@Category** et la valeur par défaut pour le paramètre **@DateToCheck**. Vérifiez les résultats.<br>3. Testez la procédure stockée **Sales.GetDiscountsForCategoryAndDate** en utilisant **'Reseller'** comme valeur de **@Category** et le mois suivant le mois en cours comme valeur de **@DateToCheck**. Vérifiez les résultats. |

| Tâche | Informations |
|---|---|
| Créer et tester la procédure stockée **AddDiscount**. | 1. Créez la procédure stockée **Sales.AddDiscount** conformément aux impératifs, puis exécutez votre instruction CREATE PROCEDURE, en contrôlant qu'aucune erreur ne se produit.<br>2. Testez la procédure stockée **Sales.AddDiscount** en utilisant les valeurs de paramètre suivantes :<br>**Parameter** — **Value**<br>@Description — 'Moitié prix pour tout'<br>@DiscountPct — .5<br>@Type — 'Remise<br>@Category — 'Client'<br>@StartDate — Local variable with current date from the GETDATE function<br>@EndDate — Local variable with date one month from current date<br>@MinQty0 — @MaxQty20<br>3. Confirmez qu'une nouvelle offre spéciale a été créée en affichant la valeur du paramètre de sortie **@NewProductID**.<br>4. Copiez le test précédent et modifiez la valeur de **@DiscountPct** en **– 0,5**. Utilisez la valeur de retour de la procédure stockée pour déterminer si l'insertion a réussi ou échoué. Si l'insertion a échoué, sélectionnez l'enregistrement le plus récent de la table **dbo.ErrorLog** pour confirmer l'erreur. |

| Tâche | Informations |
|---|---|
| Modifier le contexte d'exécution de la procédure stockée **AddDiscount**. | 1. Modifiez la procédure stockée **Sales.AddDiscount** pour qu'elle s'exécute en tant que **Michel**, puis exécutez votre instruction ALTER PROCEDURE, en contrôlant qu'aucune erreur ne se produit.<br>2. Testez la procédure stockée **Sales.AddDiscount** en utilisant les valeurs de paramètre suivantes :<br>**Parameter** — **Value**<br>@Description — 'Moitié prix pour tout'<br>@DiscountPct — -0.5<br>@Type — 'Remise'<br>@Category — 'Client'<br>@StartDate — Local variable with current date from GETDATE function<br>@EndDate — Local variable with date one month from current date<br>@MinQty — 0<br>@MaxQty — 20<br>3. Lorsque l'insertion échoue, sélectionnez l'enregistrement le plus récent de la table **dbo.ErrorLog** pour confirmer que le **UserName** (nom d'utilisateur) est maintenant enregistré en tant que **Michel** au lieu du compte **dbo**. |

## 3. Exercice 2 : Création de fonction

# Exercice 2 : Création de fonctions

**Création des fonctions GetMaximumDiscount ForCategory, GetDiscountsForDate et GetDiscountedProducts**

| Tâche | Informations |
|---|---|
| Créer et tester la fonction **GetMaximumDiscount ForCategory** définie par l'utilisateur. | 1. Ouvrez le fichier de requête **UserDefinedFunctions.sql**.<br>2. Créez la fonction scalaire **Sales.GetMaximumDiscountForCategory** conformément aux impératifs, puis exécutez votre instruction CREATE PROCEDURE, en contrôlant qu'aucune erreur ne se produit.<br>3. Testez la fonction scalaire **Sales.GetMaximumDiscountForCategory**, puis vérifiez les résultats. |
| Créer et tester la fonction **GetDiscountsForDate** définie par l'utilisateur. | 1. Créez la fonction table incluse **Sales.GetDiscountsForDate** conformément aux impératifs, puis exécutez votre instruction CREATE FUNCTION, en contrôlant qu'aucune erreur ne se produit.<br>2. Testez la fonction table incluse **Sales.GetDiscountsForDate**, puis vérifiez les résultats. |
| Créer et tester la fonction **GetDiscountedProducts** définie par l'utilisateur. | 1. Créez la fonction table incluse à instructions multiples **Sales.GetDiscountsForDate** conformément aux impératifs, puis exécutez votre instruction CREATE FUNCTION, en contrôlant qu'aucune erreur ne se produit.<br>2. Testez la fonction table à instructions multiples **Sales.GetDiscountedProducts**, puis vérifiez les résultats. |

**Liste de contrôle des résultats**

Utilisez la liste suivante de contrôle des résultats pour vérifier si vous avez exécuté cet atelier pratique avec succès :

- Création d'une procédure stockée nommée **Sales.GetDiscounts**
- Création d'une procédure stockée nommée **Sales.GetDiscountsForCategory**
- Création d'une procédure stockée nommée **Sales.GetDiscountsForCategoryAndDate**
- Création d'une procédure stockée nommée **Sales.AddDiscount**
- Création d'une fonction définie par l'utilisateur nommée **Sales.GetMaximumDiscountForCategory**
- Création d'une fonction définie par l'utilisateur nommée **Sales.GetDiscountsForDate**
- Création d'une fonction définie par l'utilisateur nommée **Sales.GetDiscountedProducts**