



Dotnet France  
Technologies Sharepoint, SQL Server & .NET

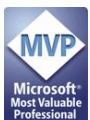
Association Dotnet France

# Les vues et les index

*Version 1.0*



Grégory CASANOVA



James RAVAILLE

<http://blogs.dotnet-france.com/jamesr>

# Sommaire

---

1	Introduction.....	3
1.1	Présentation .....	3
1.2	Pré-requis .....	3
2	Présentation des vues .....	4
2.1	Généralités sur les vues.....	4
2.2	Création d'une vue .....	4
2.2.1	Avec du code T-SQL.....	4
2.2.2	Avec SSMS .....	6
2.3	Suppression d'une vue .....	9
2.3.1	Avec du code T-SQL.....	9
2.3.2	Avec SSMS .....	9
3	Les index.....	11
3.1	Généralités sur les index .....	11
3.2	Index ordonnés ou non ?.....	11
3.2.1	Les index organisés (index cluster).....	11
3.2.2	Les index non organisés(index non-cluster).....	12
3.3	Créer un index .....	12
3.4	Supprimer un index .....	13
3.5	Reconstruire un index .....	13
3.6	Mettre à jour les statistiques .....	14
4	Conclusion .....	17

# 1 Introduction

## 1.1 Présentation

Dans les chapitres précédents, en particulier les chapitres 2 et 3, nous avons présenté les bases de données SQL Server, ainsi que les objets qu'elles peuvent contenir. On appelle objet d'une base de données, toute entité qui peut être comprise dans une base de données.

Nous allons continuer de présenter ces objets dans ce chapitre, et introduisant et en approfondissant les notions concernant les vues et d'index dans une base de données, en présentant quels sont leurs buts principaux, et comment les gérer.

## 1.2 Pré-requis

Pour une bonne compréhension de ce chapitre, vous aurez besoin au préalable d'avoir lu les sujets suivants :

- La manipulation de SSMS et de ses fonctionnalités (Chapitre 1).
- Connaitre les généralités sur les bases de données relationnelles (Chapitre 2).
- Maîtriser la plupart des actions possibles sur des objets de la base, en tout cas celles que nous avons vues (Chapitre 3 et 4). Il est important de noter que nous avons défini ces « actions » comme étant des actions du DDL dans le chapitre précédent.

## 2 Présentation des vues

### 2.1 Généralités sur les vues

SQL Server 2008 permet la gestion d'objets associés aux tables, les vues. On peut définir une vue comme étant une table dite *virtuelle*, qui a la même utilisation qu'une table, simplement une vue ne prend pas d'espace sur le disque, puisqu'elle ne stocke pas les données comme une table. Elle ne stocke que la requête d'extraction des données (SELECT).

Les vues sont un grand avantage quand à la gestion des données, vis-à-vis de l'utilisateur final. En effet, elles permettent tout d'abord de simplifier la structure des tables, qui peuvent parfois comporter une multitude de colonnes. On pourra alors choisir, en fonction de l'utilisateur, les colonnes dont il aura besoin, et n'inclure que ces colonnes dans notre vue. Une vue peut aussi permettre la réutilisation des requêtes. En effet, lorsque certaines requêtes sont souvent utilisées, une vue permettra de stocker cette requête et de l'utiliser plus facilement. Et bien entendu, l'atout majeur des vues : la sécurité d'accès aux données. Il sera possible au travers de vues, de ne donner accès à un ou des utilisateurs particuliers, que les colonnes d'une table que nous voulons qu'il voie, et pas les autres. Ce contrôle se fait grâce à la gestion des droits d'utilisateurs et de groupes (Ce sujet est traité dans la partie Administration de SQL Server 2008). Comme tout les autres objets d'une base de données, on peut créer une vue de deux manières. Par l'interface graphique, ou bien par des instructions T-SQL dans le registre DDL. Les vues proposent donc des avantages quant à leur création :

- **Simplification** de la structure des tables.
- **Réutilisation** des requêtes.
- **Sécurité** d'accès.

**Important** : La quasi-totalité des instructions du DML sont applicables sur les vues (INSERT, UPDATE, DELETE). Il faut simplement faire attention lors de l'insertion aux colonnes n'acceptant pas les valeurs NULL. De plus, l'insertion de données au travers d'une vue agrégeant des colonnes provenant de plusieurs tables est impossible.

### 2.2 Création d'une vue

#### 2.2.1 Avec du code T-SQL

La syntaxe de création d'une vue avec du code T-SQL est simple. On utilisera l'instruction `CREATE`, comme pour toute création d'objet dans une base de données.

```
CREATE VIEW Nom_Vue  
[options1]  
AS requête [options2]
```

Nous utilisons l'instruction `CREATE VIEW`, auquel nous associons le nom que nous voulons lui donner. Le mot clé `AS` indique que nous allons spécifier la requête `SELECT` qui va nous permettre de sélectionner les colonnes d'une ou plusieurs tables, afin d'en copier les propriétés dans la vue que nous créons. Il est bon de préciser que des clauses existantes pour une instruction `SELECT` classique ne conviendra pas pour une instruction `SELECT` servant à créer nos vues. Ces instructions ne doivent pas être autre que l'instruction `SELECT`, et les clauses `FROM` et `WHERE`. Concernant les options 1 et 2 : l'option 1 correspond aux options suivantes : `WITH ENCRYPTION`, `WITH SCHEMABINDING`, `WITH`

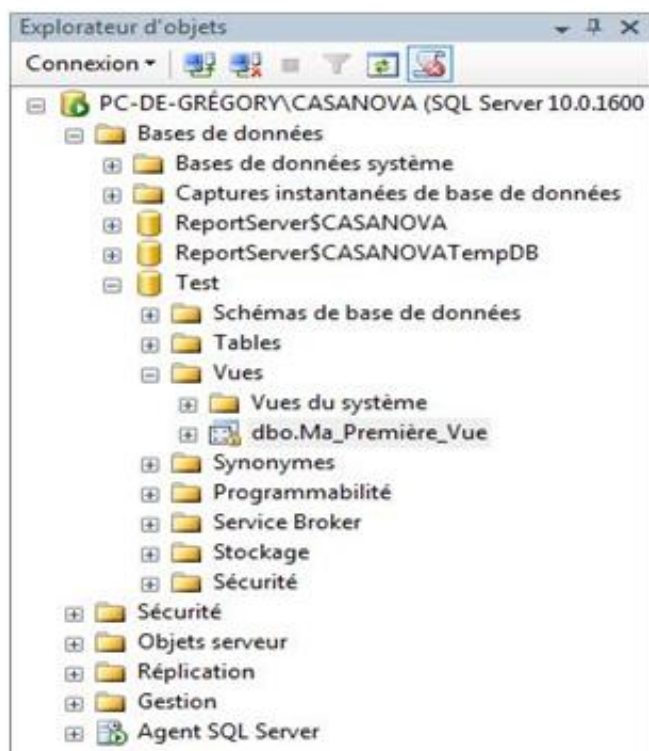
VIEW\_METADATA ; et l'option 2 correspond à l'option suivant : WITH WHECK OPTION. Découvrons les actions de chacune des ces options sur notre vue.

- **WITH ENCRYPTION** : permet de crypter le code dans les tables système. Attention : personne ne peut consulter le code de la vue, même pas son créateur. Lors de la modification de la vue avec l'instruction ALTER VIEW, il sera nécessaire de préciser de nouveau cette option pour continuer à protéger le code de la vue.
- **WITH SCHEMABINDING** : permet de lier la vue au schéma. Avec cette option, il est impératif de nommer nos objets de la façon suivante : schéma.objet.
- **WITH VIEW\_METADATA** : permet de demander à SQL Server de renvoyer les métadonnées correspondantes à la vue, et non celles qui composent la vue.
- **WITH WHECK OPTION** : permet de ne pas autoriser l'insertion ni la modification des données ne correspondant pas aux critères de la requête.

Voici un exemple :

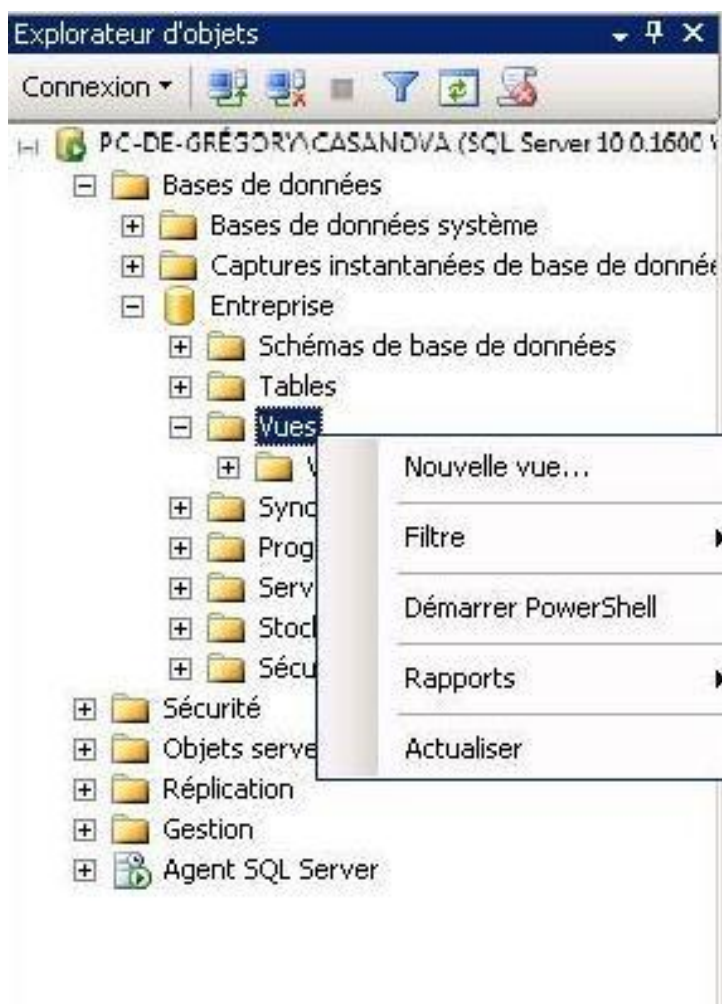
```
CREATE VIEW Ma_Premiere_Vue
WITH ENCRYPTION
AS SELECT Id_Client, Nom_Client
FROM dbo.Client
```

Cet exemple permet de créer une vue dont le nom est Ma\_Première\_Vue, avec l'option WITH ENCRYPTION, et cette vue contiendra les colonnes Id\_Client\_archive et Id\_Commande\_archive de la table Archive. Vous pourrez alors retrouver votre vue dans le sous dossier de votre explorateur d'objet, comme présenté dans l'image ci-dessous :

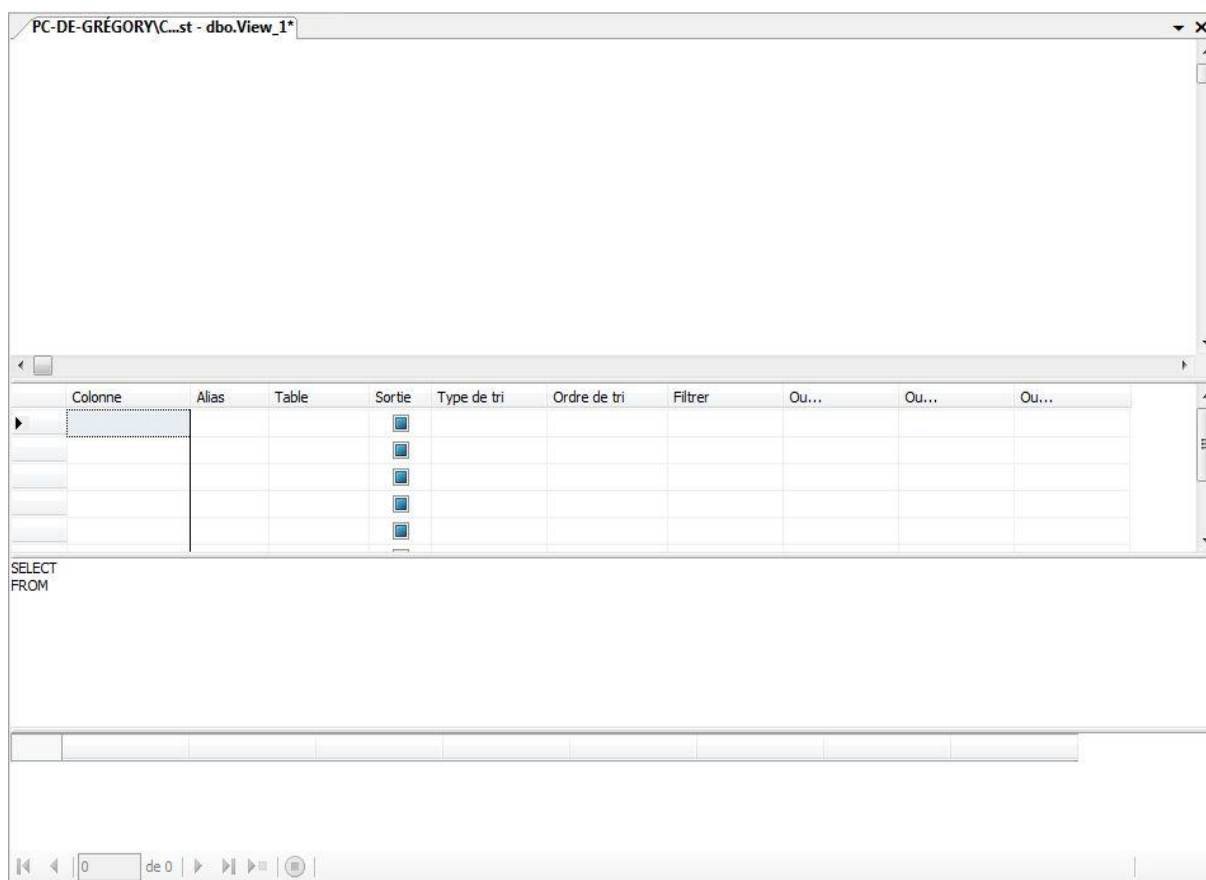
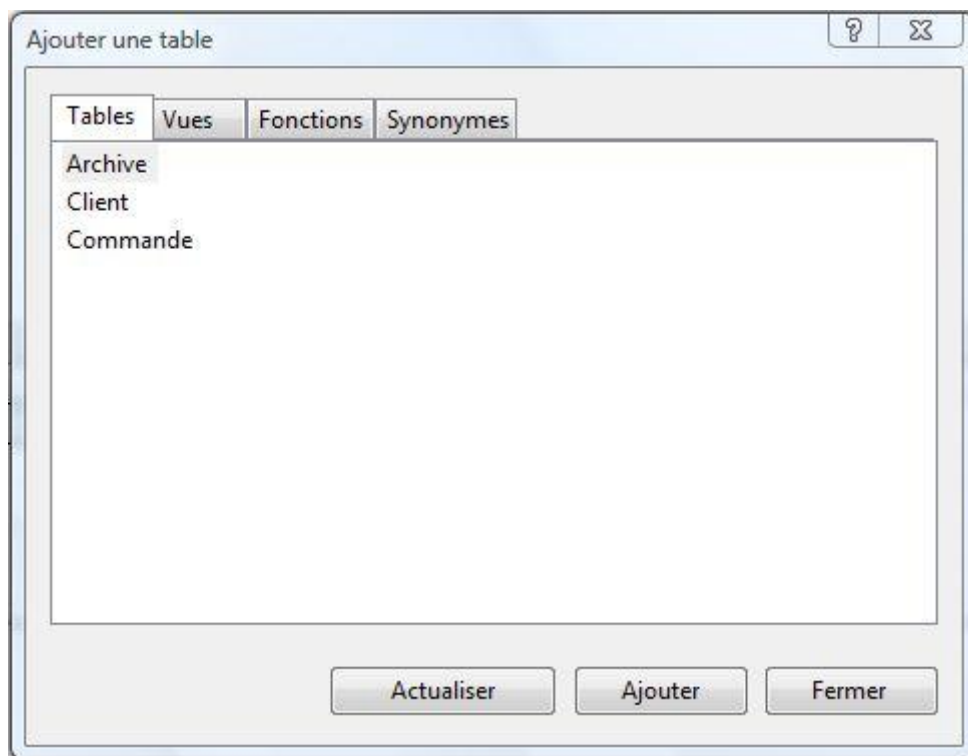


### 2.2.2 Avec SSMS

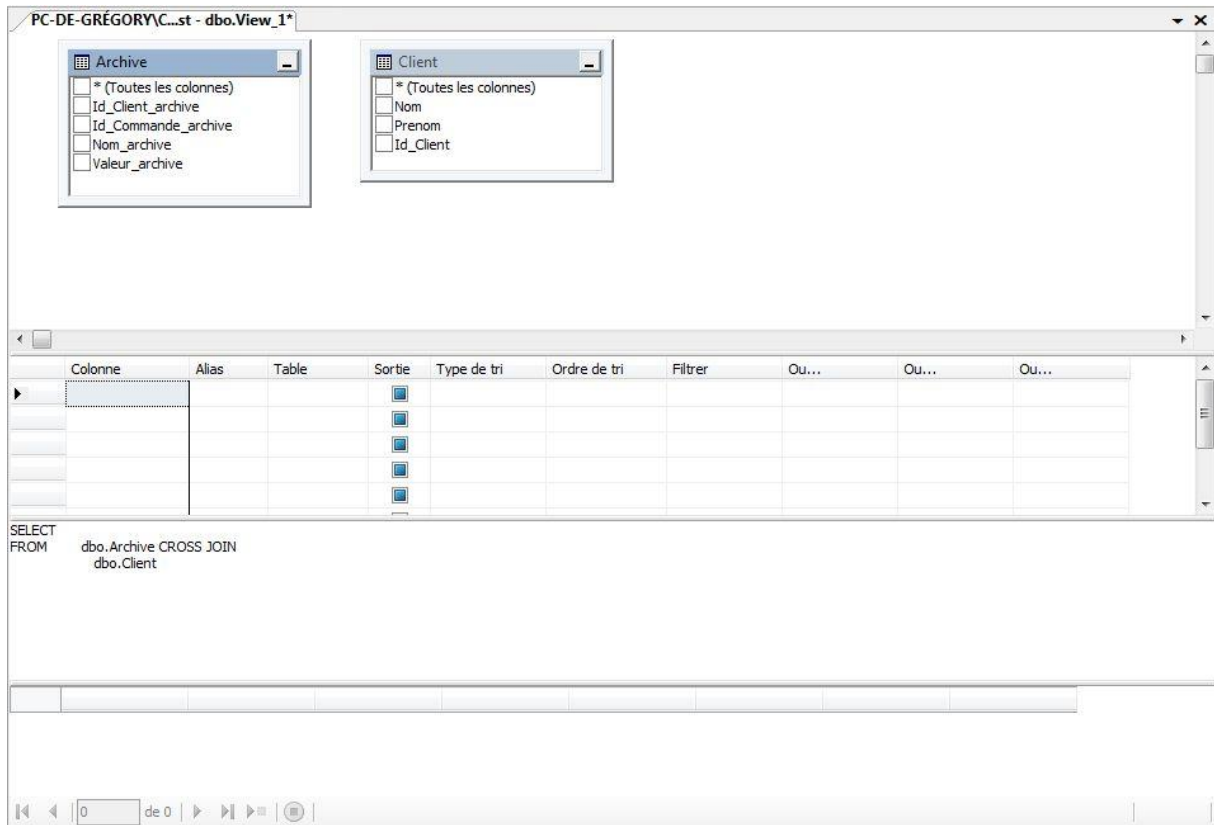
Il est très intuitif de créer une vue grâce à SSMS. Pour ce faire, de manière graphique, il vous suffit d'effectuer un click droit sur le sous dossier « Vues » dans votre base de données, affichée dans l'explorateur d'objet. Après avoir effectué le click droit, sélectionnez « Nouvelle vue... »



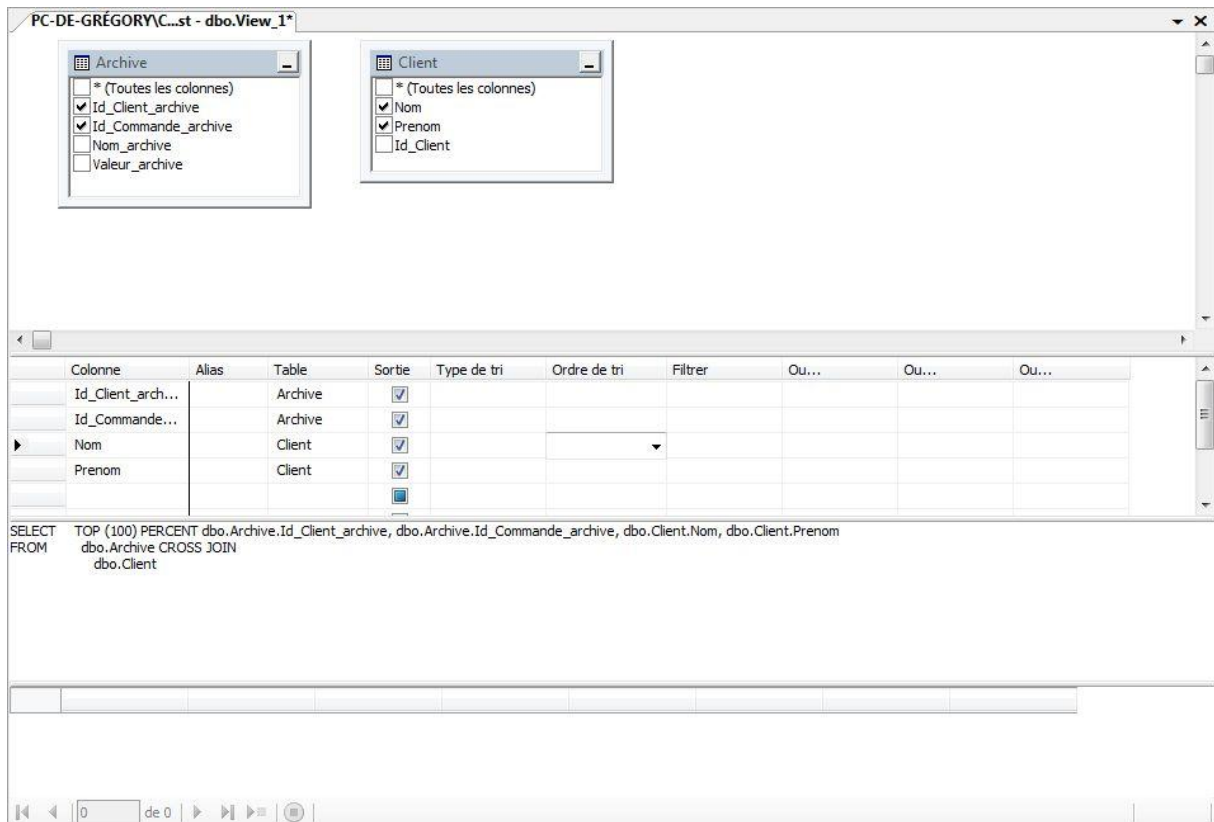
Après avoir cliqué sur “Nouvelle vue...”, les deux fenêtres suivantes apparaissent au sein même de SSMS. La première vous aidera à sélectionner des tables sur lesquelles la vue portera. La seconde vous permet de sélectionner les colonnes à utiliser et construire votre requête.



Pour ajouter une table, cliquez sur le nom de la table voulue, et sélectionnez « Ajouter ». Vous pouvez en ajouter plusieurs, via l'utilisation de la touche [Control]. On remarquera que les tables, aussitôt sélectionnées, sont modélisées dans la partie supérieure de la seconde fenêtre.



Pour sélectionner les colonnes à mettre dans votre vue, cochez les cases correspondant à vos colonnes dans les tables modélisées dans la partie supérieure de la fenêtre. Lorsque l'on coche des cases, on peut remarquer que le nom de ses colonnes est ajouté dans la partie centrale de la fenêtre.



On peut alors modifier le type de tri, l'ordre, ou encore le filtre de cette vue en changeant les caractéristiques dans la même partie. Toutes les actions effectuées permettront de générer le code de la vue. Voici un exemple :

```
SELECT TOP (100) PERCENT dbo.Archive.Id_Client_archive, dbo.Archive.Id_Commande_archive, dbo.Client.Nom, dbo.Client.Prenom
FROM   dbo.Archive CROSS JOIN
       dbo.Client
```

Lorsque vous avez fini de concevoir votre vue, faites un clic droit sur l'onglet de la fenêtre dans SSMS, et choisissez « Enregistrer ». Donnez-lui un nom. Vous venez de créer votre vue.

## 2.3 Suppression d'une vue

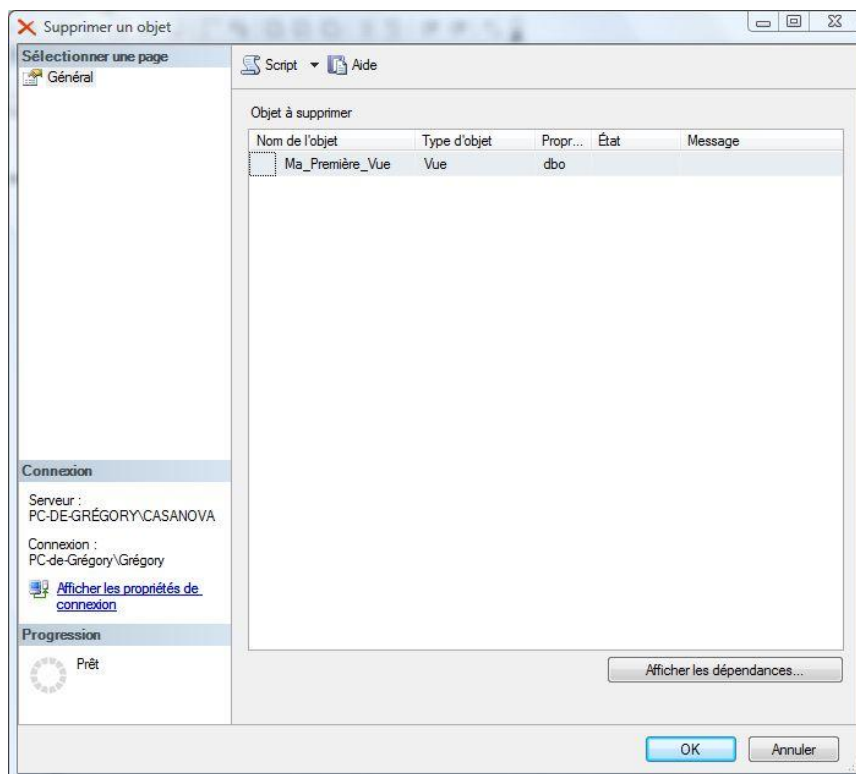
### 2.3.1 Avec du code T-SQL

La structure de suppression d'une vue est la même que pour tout objet de la base de données. Elle est la suivante :

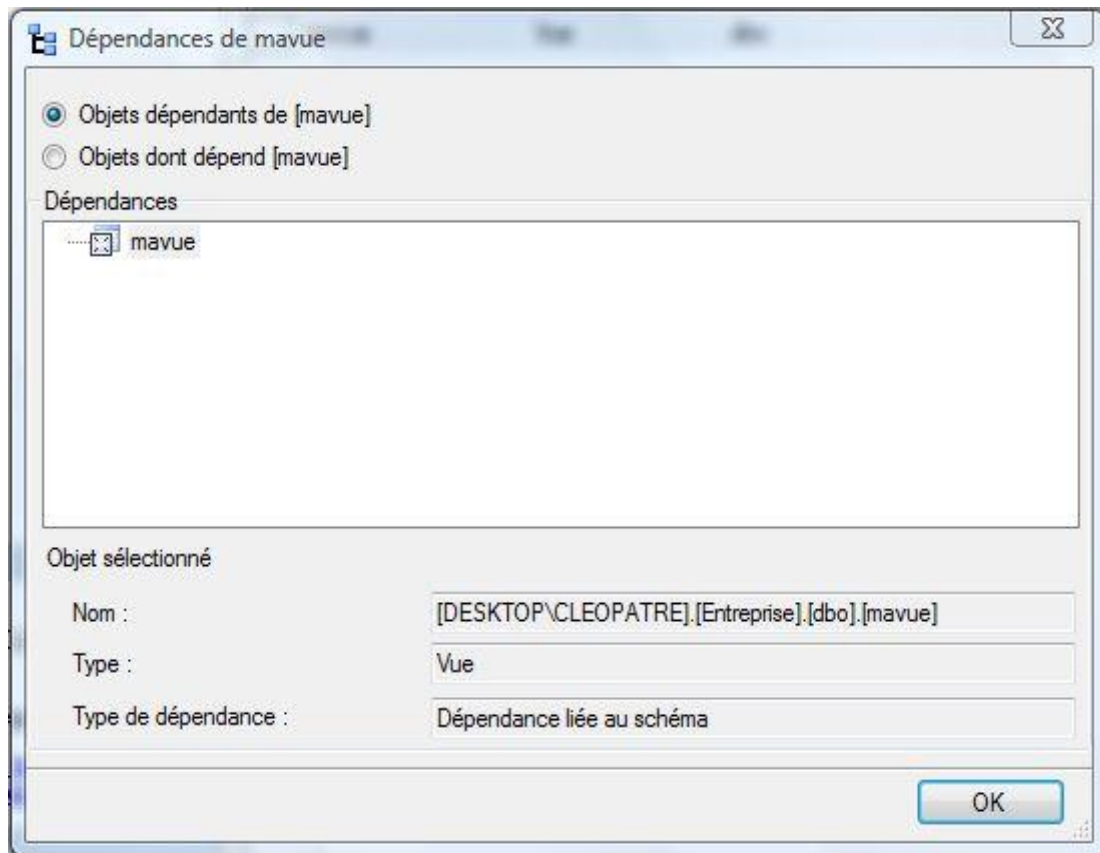
```
DROP VIEW nom_vue
```

### 2.3.2 Avec SSMS

Avec SSMS, il vous suffit d'effectuer un clic droit sur la vue dans l'explorateur d'objet et de sélectionner « Supprimer » et de cliquer sur « ok » dans la nouvelle fenêtre qui apparaît.



Grâce au bouton, « Afficher les Dépendances... », Il est possible de mettre en évidence les dépendances existantes entre les objets de la base et la vue sélectionnée. Une nouvelle fenêtre s'affiche à l'écran, dans laquelle vous pourrez choisir les différents types de dépendances.



Il vous suffit juste de savoir si vous voulez afficher les objets dépendants de la vue ou ceux dont dépend la vue en question.

## 3 Les index

### 3.1 Généralités sur les index

Les index que nous allons présenter dans ce chapitre, sont similaires aux index qui sont présents dans un livre par exemple, ou sur un site web. Si le but de l'index d'un livre est de nous permettre d'accéder plus rapidement au sujet qui nous intéresse dans ce livre, il en est de même pour les index dans la base de données, à la différence que ces index vont nous permettre de retrouver plus rapidement les données stockées dans la base. Si les index permettent de retrouver des données plus rapidement dans la base de données, ils ont un inconvénient majeur : ils sont coûteux en mémoire et espace disque, surtout dans le cas de mises à jours de colonnes indexées. Il faut bien garder en tête que les données visées par l'index y sont contenues, et donc que ces données occupent un espace disque non négligeable. Il est donc nécessaire de bien définir la stratégie d'indexation à adopter en fonction des raisons évoquées plus haut, desquelles on peut établir deux règles importantes :

- Il est mieux d'avoir trop peu d'index que trop. Imaginez qu'un index existe pour chaque page d'une livre. Il reviendra au même de feuilleter le livre page à page pour trouver ce que l'on cherche.
- Les index doivent être le plus large possible afin de pouvoir être utilisés par plusieurs requêtes.

Il existe plusieurs types d'index (index ou index-cluster, unique ou non unique...). Dans ce cours nous mettrons davantage l'accent sur leur création, mise à jour et suppression. Nous ne traiterons pas des index XML, car leur indexation est particulière. Nous le traiterons dans un chapitre propre au traitement des données de type XML. De plus, le partitionnement d'index est possible, mais c'est une solution assez avancée en termes de difficulté. Nous le traiterons dans un chapitre ultérieur.

**Remarque :** il faut bien garder en tête que les index sont les catalyseurs de nos requêtes. Ils vont permettre sur un serveur de plus grande envergure, d'obtenir des temps de réponse de plus en plus courts, du fait que la requête n'aura pas à accéder et à parcourir la totalité des données dans les tables, mais accèdera à un nombre beaucoup moins conséquent de données dans notre index.

### 3.2 Index ordonnés ou non ?

Sql server propose deux types d'index pour ses bases de données, les index organisés ou ordonnés (un seul par table), et les index non organisés ou non ordonnés (aucun, un ou plusieurs par table). Chaque type d'index possède ses avantages et ses inconvénients. Prenons le temps d'expliquer les caractéristiques de chacun d'eux.

#### 3.2.1 Les index organisés (index cluster)

Sur chaque table de votre base de données, il n'est possible de créer qu'un et un seul index organisé. Ce type d'index permet d'organiser physiquement les données d'une table en fonction d'une colonne. Pour prendre un exemple, une table qui possède une clé primaire possède aussi un index ordonné, c'est pourquoi les données sont rangées en fonction de la colonne où est définie la clé

primaire. Ce type d'index est coûteux en temps et espace disque pour le serveur lors de la construction ou la reconstruction de celui-ci. Il est bon de noter que si des index non ordonnés existent déjà sur une table, la construction d'un index sera d'autant plus longue qu'il y aura un nombre conséquent d'index non ordonnés définis sur cette table. Enfin, il est bon d'éviter de définir des index ordonnés sur des champs tels que des noms ou des prénoms, car cette pratique mène irrémédiablement à la dégradation des performances du serveur.

### 3.2.2 Les index non organisés (index non-cluster)

Les index non organisés sont les index à privilégier pour couvrir plusieurs requêtes dans SQL Server. En effet, leur utilisation reste couteuse en espace disque, mais les requêtes évitent un accès inutile à la table, puisque seul l'index est manipulé. Les performances sont alors améliorées grandement, puisque la totalité des données n'est pas manipulée, mais seulement une partie des données, celle stockée dans l'index.

## 3.3 Créer un index

Un index peut être créé à n'importe quel moment, qu'il y ait ou non des données dans la table. Simplement, il est préférable de créer l'index après une importation majeure de données, pour éviter à avoir à le reconstruire par la suite, ce qui causera une perte conséquente de temps au niveau serveur. Voici la syntaxe générale de création d'un index, que celui-ci soit ordonné ou non :

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX Nom_Index
ON Nom_Table (Nom_Colonne1)
[INCLUDE (Nom_Colonne2)]
[WITH]
[PAD_INDEX OFF], [FILLFACTOR = x],
[IGNORE_DUP_KEY = OFF], [DROP_EXISTING OFF],
[ONLINE = OFF], [STATISTICS_NORECOMPUTE = OFF]
[ON Nom_Groupe_Fichier]
```

Pour créer un index, nous nous servons de l'instruction `CREATE INDEX`. Les choix compris entre les mots clés `CREATE` et `INDEX` servent à décider si l'index doit être ordonné ou non, et s'il doit être unique. Avec la clause `ON`, on définit sur quelles colonnes porte l'index et avec la clause `INCLUDE`, on passe en paramètres plus de colonnes de la même table, afin que les requêtes n'aient qu'à chercher dans l'index lorsqu'elles s'exécutent. La clause `ON` en toute fin de lot permet de définir sur quel groupe de fichier nous allons placer notre index. On peut noter que si le groupe de fichier n'est pas précisé, l'index sera placé dans le groupe de fichier principal. A la suite de la clause `WITH`, nous trouvons les différentes options des index, qui par défaut sont toutes désactivées (OFF) mis à part `FILLFACTOR`. Voici le détail des services que proposent ces options :

- `PAD_INDEX` : Précise le niveau de remplissage du niveau non feuille. Cette option n'est utilisable qu'avec `FILLFACTOR` dont la valeur est reprise.
- `FILLFACTOR` : Précise le pourcentage de remplissage des pages d'index au niveau feuille. La valeur par défaut est 0.
- `IGNORE_DUP_KEY` : Cette option autorise les entrées doubles dans les index de type `UNIQUE`.
- `DROP_EXISTING` : Précise que l'index existant doit être supprimé.

- **ONLINE** : Lorsque cette option est activée, les données de la tables restent accessibles en lecture, lors de la création de l'index.
- **STATISTICS\_NORECOMPUTE** : Désactivé, cette option précise que les statistiques ne seront pas mise à jour.

Nous allons maintenant créer un index non ordonné sur notre base de données Entreprise, base de données dont nous nous servons pour tous nos exemples dans ce cours. Cette base est accessible en annexe. Prenons un exemple simple. Dans notre table Client de la base de données Entreprise, le mail des client peut être à la valeur NULL. De plus, on remarque que le champ Mail\_Client n'est occupé que pour 50% des cas, soit un cas sur deux. L'accès aux données est donc ralenti par le fait que la plupart des données sont à NULL. On peu donc créer un index non organisé de cette manière :

```
CREATE NONCLUSTERED INDEX Index_Mail
ON Client (Mail_Client)
INCLUDE (Id_Client)
WHERE Mail_Client IS NOT NULL
```

Avec ce segment de code, nous allons créer un index non organisé, qui s'appelle Index\_Mail, sur la table Client, pour la colonne Mail\_Client, à laquelle nous incluons Id\_Client. Cet index va récupérer toutes les informations de la table Client pour laquelle Mail\_Client n'est pas NULL, ce qui va nous permettre d'accélérer nos requêtes de façon conséquente, dans le cas où la masse de données sur la base est conséquente.

### 3.4 Supprimer un index

Comme chacun le sait désormais, le mot clé pour supprimer un objet de la base est le mot **DROP**. Nous allons encore une fois l'utiliser afin de pouvoir supprimer un index de la base. Voici la commande type de suppression d'un index :

```
-- DROP INDEX Nom_Index ON Nom_Table

DROP INDEX Index_Mail ON Client
```

La suppression d'index peut avoir plusieurs origines. La plus fréquente est la suivante. Lorsqu'un index est trop couteux en maintenance et qu'il n'offre pas de performances significatives sur les requêtes, il peut être préférable de le supprimer.

### 3.5 Reconstruire un index

Autrefois, sur les versions antérieures à SQL Server 2008, la commande de reconstruction d'index était la suivante : DBCC DBREINDEX. Pour des soucis de compatibilité, cette instruction est maintenue, mais il est préférable de ne plus l'utiliser dans le cadre d'un nouveau développement. Maintenant, il est possible de reconstruire un index avec l'instruction **ALTER INDEX**. Cette instruction va nous permettre de reconstruire un index particulier, ou tous les index d'une table en particulier. Lors de cette instruction, nous pouvons changer les caractéristiques de l'index que nous avons pu proposer lors de sa construction (**FILLFACTOR**, **PAD\_INDEX**...). La syntaxe de reconstruction est la suivante :

```
ALTER INDEX ( Nom_Index | ALL )
ON Nom_Table
REBUILD
[WITH]
[PAD_INDEX OFF], [FILLFACTOR = x],
[IGNORE_DUP_KEY = OFF], [DROP_EXISTING OFF],
[ONLINE = OFF], [STATISTICS_NORECOMPUTE = OFF]
[ON Nom_Groupe_Fichier]
```

A la suite de l'instruction `ALTER INDEX`, il est nécessaire de préciser le nom du ou des index à reconstruire, ou bien de préciser si l'on veut que tous les index soient reconstruits en précisant le mot clé `ALL`. La clause `ON` permet de préciser de quelle table sont originaires les éventuels index à reconstruire et le mot clé `REBUILD` permet de préciser que l'on veut reconstruire ces index. La dernière clause `WITH` permet quant à elle de préciser les caractéristiques des index à reconstruire. Toutes les options contenues dans le `WITH` fonctionnent de la même manière que pour la simple construction de l'index.

### 3.6 Mettre à jour les statistiques

SQL Server utilise des informations statistiques pour optimiser les requêtes. Ces informations doivent être mises à jour avant une modification importante des données. Deux méthodes de mise à jour de ces informations sont possibles : la manuelle et l'automatique. En revanche, il est fortement conseillé d'utiliser les fonctions automatiques de mise à jour de statistiques, pour une unique raison. Dans la plupart des cas, la dégradation des performances au niveau serveur est causée majoritairement ou en totalité par le fait que ces mises à jour statistiques n'ont pas été faites. Nous allons donc présenter la manière à employer pour les mettre à jour manuellement, simplement, il est fortement conseillé de les mettre à jour avec le système automatique. La syntaxe de mise à jour des statistiques est la suivante :

```
UPDATE STATICS Nom_Table Nom_Indexs
WITH (FULLSCAN | SAMPLE n (PERCENT | ROWS) | RESAMPLE)
```

Pour mettre à jour les statistiques, la syntaxe présentée est la bonne. `UPDATE STATICS` nous permet d'annoncer que nous allons mettre à jour les statistiques. Il est alors nécessaire de préciser le nom de la table concernée, et le nom des index dont les statistiques doivent être mise à jour. Si le nom des index est omis, toutes les statistiques de tous les index seront mis à jour. La clause `WITH` permet de préciser les modes de mise à jour que nous allons utiliser. Détaillons les options possibles :

- `FULLSCAN` : Les statistiques vont être créées avec un balayage complet de la table, soit 100% des lignes contenues dans la table précisée en paramètre.
- `SAMPLE` : Les statistiques vont être établies à partir d'un échantillon représentatif des informations contenues dans la table précisée en paramètre. La valeur `n` représente la valeur que nous voulons des lignes à prendre en compte pour notre échantillon, quand aux mots clés `PERCENT` et `ROWS`, ils permettent de savoir dans l'ordre, si la valeur de `n` comprend une valeur en pourcentage ou une valeur en nombre de lignes de la table indiquée.
- `RESAMPLE` : Permet de redéfinir les statistiques à partir d'un nouvel échantillonnage.

Comme dit précédemment, la mise à jour des statistiques de manière manuelle est à proscrire, dans le sens où les mises à jour automatiques sont possibles et que les performances de notre serveur en dépendent. Voyons de quelle manière nous allons opérer pour permettre à nos bases de mettre à jour automatiquement les statistiques de nos index. Deux manières existent pour arriver à nos fins. Par l'instruction ALTER DATABASE ou par la procédure stockée système sp\_autostats. Il faut simplement analyser les besoins que nous avons. Voyons avec les deux options possibles :

- Avec ALTER DATABASE :

```
ALTER DATABASE Entreprise
SET AUTO_CREATE_STATISTICS ON
```

Il suffit de préciser le nom de la base après l'instruction et de mettre à ON l'option AUTO\_CREATE\_STATISTICS. Ceci permet la mise à jour automatique des statistiques.

- Avec sp\_autostats :

```
exec sp_autostats Client
```

L'utilisation de la procédure stockée sp\_autostats nécessite trois arguments maximum dont un seul obligatoire, le nom de la table sur laquelle nous voulons appliquer la mise à jour automatique des statistiques.

### 3.7 Les vues indexées

Les vues indexées ont un unique objectif, comme tout autre objet indexé dans la base de données : améliorer les performances de nos requêtes. Celles-ci sont sûrement les objets de la base offrant le plus de gain de performance dans SQL Server. On se sert le plus souvent de ce type de vues, sur des bases de données OLAP, c'est-à-dire pour des données qui vont être le plus souvent en lecture, et très peu mises à jour. Ces index sont en particulier pratiques pour des requêtes nécessitant des jointures et des agrégations. La vue indexée a pour effet de matérialiser les données. On prend le résultat de la requête et on le stocke dans l'index. On met ensuite à jour l'index en fonction des modifications sur la table de base. La création d'un index sur une vue ce fait de la même manière que pour une table, simplement, un index sur une vue présente de caractères propres de fonctionnement et de comportement que nous avons commencé à présenter. Pour une version Entreprise de SQL Server, dès lors que votre index est créé, le moteur de base de données utilisera celui-ci. En revanche, pour les autres versions de SQL Server, il faut préciser si l'on veut utiliser l'index. La méthode est la suivante :

```
SELECT * FROM vue_Client WITH(NOEXPAND)
```

La clause WITH(NOEXPAND) spécifie qu'aucune vue indexée n'est étendue pour permettre d'accéder aux tables sous-jacentes lorsque l'optimiseur de requête traite la requête. L'optimiseur de requête traite la vue comme une table avec un index cluster. NOEXPAND s'applique uniquement aux vues indexées. Des contraintes sont à noter. En effet, il n'est possible de créer un index sur une vue que si les conditions suivantes sont rassemblées :

- Lors de la création de la vue, les options ANSI\_NULLS et QUOTED IDENTIFIER doivent être sur ON.

- ANSI\_NULL doit être ON lors de la création de toutes les tables référencées dans la vue sur laquelle sera créé l'index.
- La vue ne doit pas faire référence à d'autres vues.
- Toutes les tables référencées doivent appartenir à la même base et au même propriétaire.
- La vue doit être créée avec l'option WITH SCHEMABINDING.

Le premier index créé sur une vue doit être de type cluster unique. Par la suite il est possible de construire des index non cluster.

Pour faire un résumé rapide, les vues indexées sont pratiques et efficaces dans le cas où une vue possède une quantité remarquable de jointures et d'agrégations, et que ces données sont surtout en lecture sur la base. Enfin, pour une création sans erreurs de l'index, certaines conditions doivent être respectées.

## 4 Conclusion

Dans ce chapitre, nous avons vu deux objets supplémentaires dans la base : les vues et les index. Retenez principalement qu'une vue est une table virtuelle qui ne stocke pas les données mais les requêtes d'extractions associées, et que les index peuvent appartenir à deux catégories différentes, ordonnés et non ordonnés, et que ceux-ci servent à optimiser le temps de réponse du serveur lors de l'exécution de requêtes. Dans le chapitre suivant, nous verrons les notions de procédures stockées et de fonctions utilisateurs, largement utilisées dans SQL Server, puisqu'elles permettent une grande programmabilité et un gain de performance important.