



Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

Encryptions, compression et partitionnement des données

Version 1.0



Grégory CASANOVA

Sommaire

1	Introduction.....	3
2	Encryption transparente des données.....	4
2.1	Principe d'encryptions transparente des données	4
2.2	Mise en œuvre.....	4
3	Compression des données	6
3.1	Principe de compression des données.....	6
3.2	Mise en place.....	6
4	Partitionnement des données.....	8
4.1	Créer une fonction de partitionnement.....	8
4.2	Création du schéma de partitionnement	9
4.3	Table partitionnée	10
4.4	Indexes partitionnés.....	11
5	Conclusion	12

1 Introduction

Dans une base de données, l'administrateur se pose plusieurs questions pratiques, et notamment les suivantes : « La sécurité de mes données est-elle suffisante, de façon à ce qu'elles ne soient pas corrompues ou bien, utilisées à mauvais escient ? » « La place que mes données occupent sur le disque est trop importante, comment réduire cet espace ? » « Comment éviter que le retour de mes données dans mes applications clientes soit de plus en plus lent, et comment améliorer la montée en charge de mes données ? »

Bien entendu, il n'existe pas la sécurité parfaite, et la sauvegarde d'espace de stockage idéale, mais il existe deux procédés qui vont nous aider dans cette optique :

- L'encryption transparente de données.
- La compression des données.

De plus, l'optimisation des retours de données, ou bien la meilleure montée en charge des données peut être mise en place par le partitionnement des tables ou des index.

Nous allons expliquer, quels sont les principes, les avantages, et comment mettre en œuvre ces trois procédés, aujourd'hui essentiel à l'administration de base de données en entreprise.

2 Encryption transparente des données.

SQL Server 2008 propose un système d'encryptions des fichiers de données et des fichiers journaux pour assurer toujours plus la sécurité des données en entreprise. L'encryptions transparente, ou TDE (Transparent Data Encryption) est dynamique, ce qui signifie qu'à chaque écriture de données sur le disque, l'encryption de ces nouvelles données sera faite automatiquement.

Mais pourquoi "Transparent"? Tout simplement parce que cette encryption des données, ne sera en aucun cas visible par l'utilisateur final des données. Les données sont encryptées à l'écriture sur le disque, et décryptées par le même procédé lors de leurs utilisations par l'utilisateur final, si toute fois celui-ci possède les droits pour les utiliser.

2.1 Principe d'encryptions transparente des données

Concrètement, nous utiliserons une clé principale afin de générer un certificat valide. Ce certificat nous servira enfin à générer, à son tour, une clé d'encryptions, que nous utiliserons pour crypter les données. La génération de la clé de cryptage par le certificat peut être faite en utilisant différents types d'algorithmes d'encryptions.

Note : L'encryptions des données concerne tous les types de données, mis à part le type FILESTREAM, qui n'est pas un type relationnel. De plus, si l'encryption des données est déclenchée sur une base de données, alors les sauvegardes de cette base de données seront cryptées avec la même clé de cryptage que la base. La création des clés principales et des certificats se font toujours en se plaçant sur la base Master.

2.2 Mise en œuvre

Comme nous l'avons dit précédemment, il faut dans un premier temps créer la clé principale. Nous utiliserons le code T-SQL Suivant :

```
USE master
GO

CREATE MASTER KEY ENCRYPTION
BY PASSWORD='BlogDotnetFrance'
```

Nous créons ensuite le certificat de la façon suivante :

```
CREATE CERTIFICATE certificat
WITH SUBJECT='Blog Dotnet France'
```

Enfin nous créons la clé d'encryptions de cette manière :

```
USE LaBaseaEncrypter
GO

CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM=TRIPLE_DES_3KEY
ENCRYPTION BY SERVER CERTIFICATE certificat
```

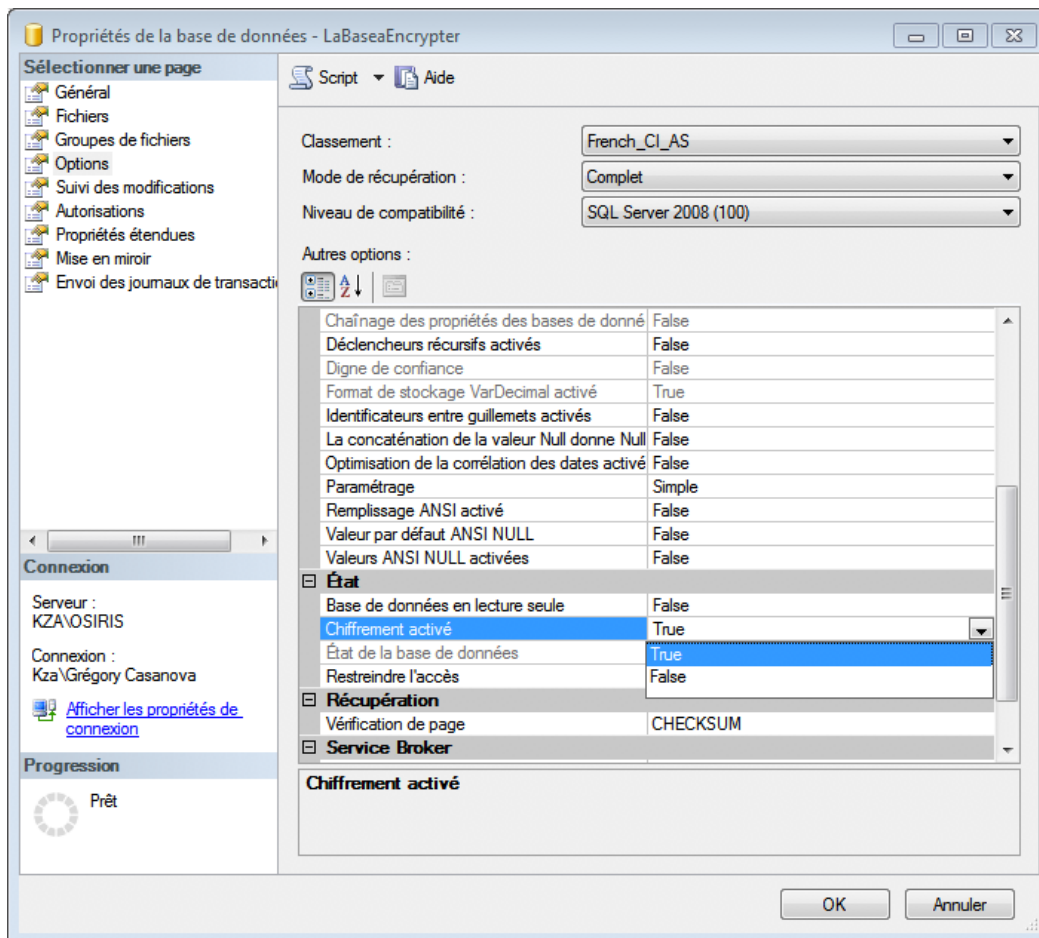
Nous somme alors prêts à encrypter les données de notre base grâce à une simple instruction ALTER DATABASE :

```
ALTER DATABASE LaBaseaEncrypter
SET ENCRYPTION ON
```

Note : Le certificat, par défaut, est perdu. Pour garder ce certificat, et par conséquent, pouvoir restaurer nos bases de données à partir de nos sauvegardes, il va falloir faire un backup de ce certificat juste après l'avoir créé. La restauration de nos bases de données se fera de la manière suivante : création d'une master key, création d'un certificat à partir du backup du certificat et restauration de la base.

L'exécution de requêtes telles que celle-ci : `SELECT * FROM sys.dm_database_encryption_keys`, vous permettra de vérifier si votre base de données est bien cryptée en vérifiant à votre tour, en fonction de l'id de la base de données, si l'encryptions state de la base est à 3. S'il est à 3, la base de données est cryptée, à 2, la base est en cour de cryptage et à 1, la base n'est pas cryptée.

Pour terminer, il est possible de rendre accessible l'encryption des données à partir de SSMS (SQL Server Management Studio), en se rendant dans la fenêtre de propriété de la base de données à crypter, catégorie Options.



3 Compression des données

Une des grandes forces de SQL Server réside dans la capacité de compression des données. Il faut savoir que certains tests en interne, chez Microsoft, ont permis de montrer que la compression des données par table et index, permet de réduire de 1,5 à 20 fois, la taille de nos données sur le disque. Lorsque la compression des données est activée sur une table ou un index, cette compression ne sera prise en compte qu'à partir de la reconstruction de la table ou de l'index touché (**ALTER TABLE nom_table REBUILD**). Dans le cas de tables partitionnées, ou d'index partitionnés, la compression des données peut être mise en place en fonction des partitions de chacune des tables ou de chacun des index.

Note : La compression des données n'est disponible que dans les versions Entreprise et Developer de SQL Server. De plus, cette compression n'est possible que pour les données utilisateur. Il n'est en aucun cas possible de compresser les données d'une table système.

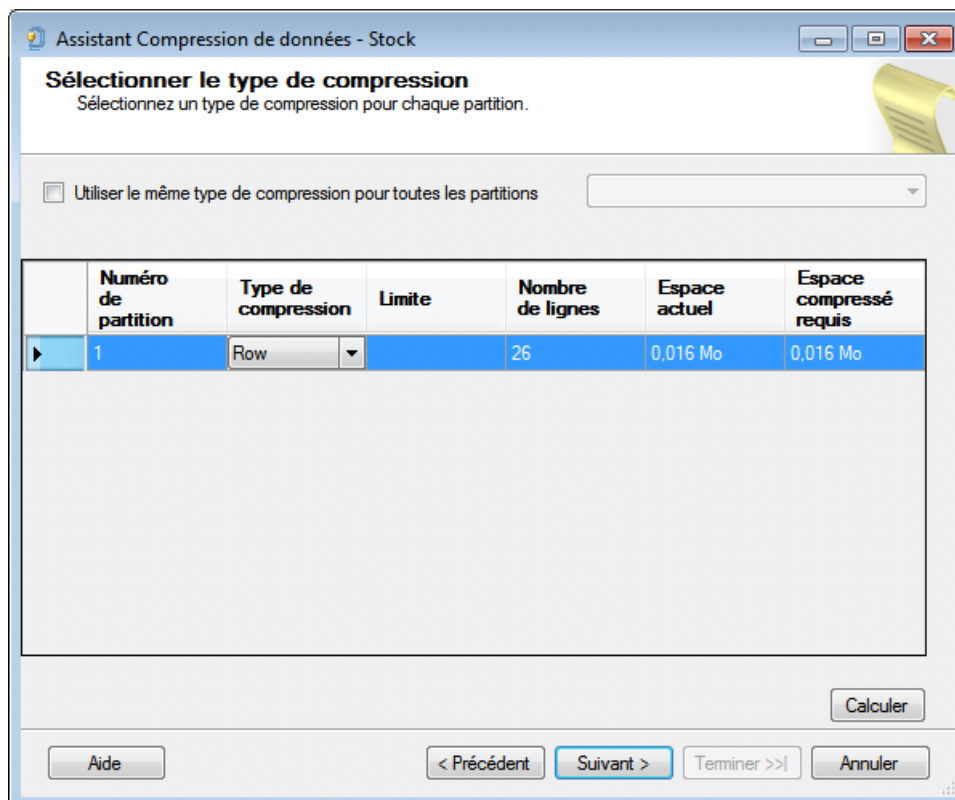
3.1 Principe de compression des données

Concrètement, la compression des données permet de réduire la taille des données sur le support de stockage, en donnant la possibilité de stocker plus de lignes de données sur un bloc de 8Ko. Si plus de ligne de données sont stockées sur un bloc de 8Ko, alors moins de blocs seront utilisés, et donc les données occuperont moins d'espace disque que dans le cas où elles ne sont pas compressées.

3.2 Mise en place

Avec du Transact-SQL, la mise en place de la compression des données se fait à la création (CREATE), ou en modifiant (ALTER) une table ou un index. La seule option à ajouter est l'option suivante : **DATA_COMPRESSION = (NONE | ROW | PAGE)** (NONE est l'option par défaut).

Avec SSMS, il existe un assistant de compression accessible de la façon suivante : Dans le menu contextuel de la table ou de l'index, choisissez l'option **Stockage**, puis **Gérer la compression**.



Dans la colonne **Type de compression**, choisissez le type de compression à utiliser (Row, Page). Il faut savoir que le choix de compression Row, va agir au niveau des lignes de la table sur laquelle la compression sera faite. En d'autres termes, la taille de chaque ligne sera affectée. En revanche, si l'on choisit l'option Page, chaque fichier de données sera compressé. On rappelle qu'un fichier de données, constitue le fichier sur lequel

sont stockée au niveau physique vos données. Ce fichier porte l'extension .mdf dans votre système de fichier Windows. Lorsque ce choix est fait, il est possible de faire une estimation de la taille actuelle des données sur le disque, et une estimation de la taille des données sur le disque après compression.

Note : Dans notre cas, la compression des données de notre table, qui ne contient que 26 lignes, ne permet pas de gagner en espace de stockage, tout simplement parce que le volume de données est trop faible, pour pouvoir réduire le nombre de pages de données. Compte tenu du fait que l'espace disque pris par les données est de 16Ko, deux pages de 8Ko sont utilisées. Il est alors simple de comprendre que les données stockées dans 2 pages de données ne peuvent pas entrer dans une page de 8Ko.

4 Partitionnement des données

Le partitionnement des données, comme énoncé plus haut, permet, et un gain de temps dans le retour des données, et une meilleure montée en charge sur le serveur, dans le cas où une multitude de requêtes est effectuée par seconde sur ce même serveur. Le partitionnement peut donc être utile pour la gestion des données, par exemple, lorsque certaines données peuvent être modifiées, alors que d'autres peuvent seulement être lues. Partitionner une table revient à diviser cette table en plusieurs sous tables, pour que chacune des requêtes effectuées sur les données de la table d'origine soit plus rapide. Le partitionnement des données d'une table permet donc d'optimiser le stockage des informations sans que le nombre de tâches administratives soit augmenté de façon significative. En théorie, SQL Server peut partitionner seul les données d'une table, seulement, ce type de partitionnement est quasiment inutile, puisqu'il n'apporte aucune valeur ajoutée à vos performances. Il est donc préférable de créer sa propre fonction de partitionnement. Pourquoi parle-t-on de fonction de partitionnement ? Tout simplement parce que sans elle, SQL Server ne saurait pas partitionner les données de la façon que nous voulons que ce soit fait. Vous l'aurez compris cette fonction de partitionnement permet donc d'indiquer comment les données seront partitionnées. Techniquement, cette fonction utilise une clé de partitionnement qui va définir la partition à utiliser en fonction du plan de partition défini.

Note : Il est possible de créer un index sur une table partitionnée, simplement, l'index sera lui aussi partitionné.

4.1 Créer une fonction de partitionnement

La fonction de partitionnement est donc celle qui va diriger les données vers l'une ou l'autre des partitions. Cette fonction utilisera des plages de valeurs pour répartir les données. On admet donc que chaque plage de valeur est bornée. Cette notion de borne est importante, suivant dans quelle partition vous voudrez placer votre valeur borne. Il est important de préciser que dans la fonction de partitionnement, nous ne préciserons que les valeurs bornes.

```
CREATE PARTITION FUNCTION nomdelafunction(int)
AS RANGE LEFT FOR VALUES (1000, 2000, 3000)
```

Dans notre cas, on crée une fonction de partitionnement de nom `nomdelafunction`, à laquelle nous passons en paramètre un `int`, le mot clé `AS RANGE` permet d'utiliser les mots clés `LEFT` ou `RIGHT`. Pour cet exemple, compte tenu qu'on utilise `LEFT`, chaque borne sera contenue dans la plage de valeur inférieure. On définit alors avec la clause `FOR VALUES` les plages de valeurs à partitionner. Dans notre cas, si X_i est une plage de valeur, elles seront les suivantes :

- $X_1 \leq 1000$
- $1000 < X_2 \leq 2000$
- $2000 < X_3 \leq 3000$
- $X_4 > 3000$

Note : Si la valeur borne doit être incluse dans la plage inférieure, on utilisera le mot clé `LEFT`, sinon, nous utiliserons le mot clé `RIGHT`. De plus, pour le partitionnement par plage de données, SQL Server trie toujours les données de la colonne cible de façon ascendante.

4.2 Création du schéma de partitionnement

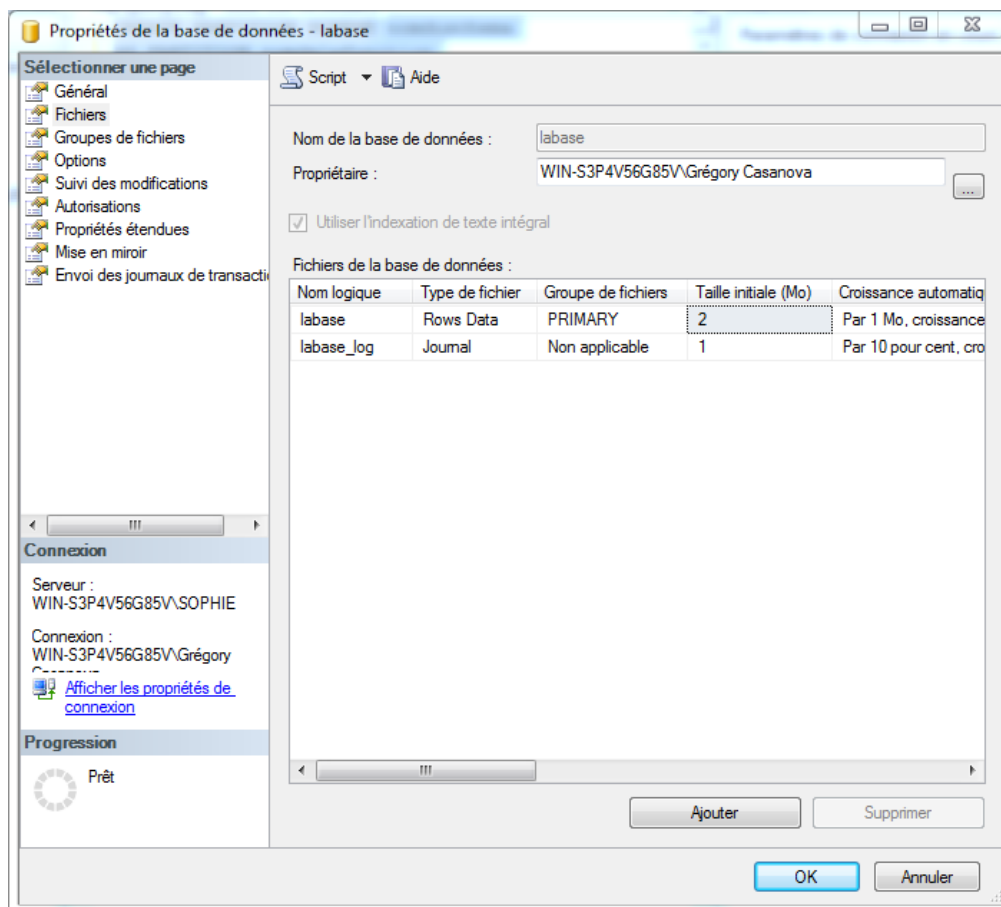
Présentons rapidement le schéma de partitionnement. Pour résumer au maximum son utilité, c'est ce schéma de partitionnement qui va faire le lien, entre les données retournées par la fonction de partitionnement et telle ou telle partition dans laquelle sera stockée telle ou telle donnée. Notons que, bien que toutes les partitions que vous utiliserez pour partitionner vos données peuvent se trouver sur le même groupe de fichier, il est recommandé de séparer chaque partition sur un groupe de fichier différent, tout simplement parce que cette pratique facilite nettement l'administration et la gestion des partitions de telle ou telle table ou index. Présentons maintenant la syntaxe de création d'un schéma de partitionnement :

```
CREATE PARTITION SCHEMA nomduschema
AS PARTITION nomdelafonction
TO (premier,seconde,troisieme)
```

Dans ce cas là, nous créons donc un schéma de partition nommé nomduschema en rapport avec la fonction de partitionnement nomdelafonction, qui utilisera les groupes de fichier, premier, seconde, et troisième.

Notons que l'utilisation de groupes de fichiers suppose que vous sachiez comment créer des fichiers de données appartenant à un certain groupe. Voici la méthode à suivre :

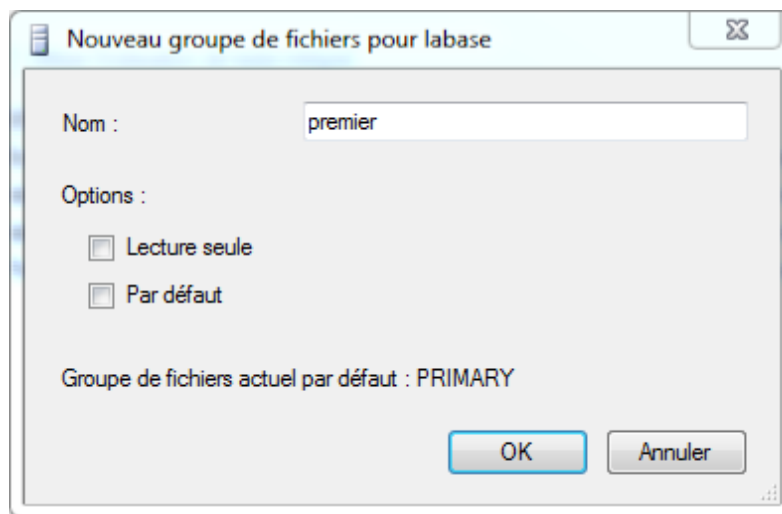
Nous allons créer trois groupes de fichiers ; premier, seconde et troisième. Rendez vous pour cela dans les propriétés de la base de données touchée. La fenêtre suivante apparaît :



Cliquez alors naturellement sur ajouter, après vous être rendu sur la partie Fichiers en haut à gauche de la fenêtre, pour ajouter un fichier de données, qui fera obligatoirement partie d'un groupe de fichier. Une troisième ligne apparaît alors, contenant différentes informations :

- Le nom logique du fichier. Ce nom sera un identifiant unique qui permettra de le reconnaître par rapport aux autres.
- Le type de fichier. Le fichier peut prendre deux types de fichiers, le type données, ou bien le type journal. Le type fichier sera celui qui nous intéressera cette fois-ci.
- Le groupe de fichier auquel il appartient. La partie qui nous intéresse le plus. En effet, lorsque un fichier de données est plein, si des fichiers existent dans le groupe de fichier, et que celui-ci est vide, les données seront stockées sur ce fichier pour éviter l'accroissement anarchique des fichiers de données de la base.
- La taille du fichier. La taille du fichier définit la taille maximale que peut prendre le fichier lorsqu'il est plein.
- L'accroissement du fichier. S'il n'existe pas d'autres fichiers dans le groupe de fichiers où l'on peut stocker les données, SQL Server gère un accroissement du fichier, pour éviter le manque de place dans le fichier de données.

Le groupe de fichiers par défaut se nomme PRIMARY. Pour créer un nouveau groupe de fichier, nous allons opérer ainsi. Dans la colonne groupe de fichiers, nous allons sélectionner nouveau groupe. Nous obtenons la fenêtre suivante :



Il vous suffit alors de lui donner un nom, et de préciser si ce fichier est en lecture seule, et s'il devient le groupe de fichier par défaut. Cliquez alors sur OK, puis une dernière fois sur OK, dans la fenêtre principale si toutefois vos opérations sont terminées.

4.3 Table partitionnée

Pour l'utilisateur final, une table partitionnée n'a aucun intérêt puisque pour lui, cette pratique n'est pas visible. En revanche, c'est un plus pour n'importe quel utilisateur. Pour partitionner une table, il vous suffit de préciser le schéma de partitionnement à utiliser lors de sa création. Voici la syntaxe :

```
CREATE TABLE matable (IdClientint)
ON nomduschema (IdClient)
```

C'est une syntaxe simple de création de table, simplement, on précise après la clause ON le schéma de partitionnement utilisé, et en paramètre à ce schéma, la colonne à utiliser pour calculer les données partitionnées. Il est important de noter que le type de valeur donné lors de la création de la fonction de partitionnement, doit être le même que le type de la colonne passée en paramètre au schéma de partitionnement.

4.4 Indexes partitionnés

Les indexes partitionnés sont aussi simples à créer que les tables partitionnées, puisqu'ils s'appuient eux aussi sur des fonction de partitionnement et des schémas de partitionnement. En revanche, il est conseillé de créer un index sur une table partitionnée plutôt que l'inverse. Sachez que la création d'un index sur une table partitionnée rend celui – ci partitionné de façon automatique. Voyons maintenant la syntaxe de la création d'un index partitionné :

```
CREATE INDEX nomdemonindexe
ON matable (IdClient)
ON nomduschema (IdClient)
```

Si toutefois vous décidez tout de même de partitionner l'index d'une table partitionnée elle même, il est préférable d'utiliser la même fonction et le même schéma de partitionnement que pour le partitionnement de la table.

5 Conclusion

Dans ce chapitre, nous avons appris trois savoirs essentiels pour un administrateur de base de données. Le premier de savoir économiser un maximum son espace disque, le second de pouvoir toujours plus sauver l'intégrité de ses données, et le troisième, de garantir un accès aux données rapidement. Ces compétences sont possibles, dans l'ordre, grâce à la compression des données, à leur encryption et au partitionnement des tables et index.