



Concepteur Développeur en Informatique

Assurer la persistance des données

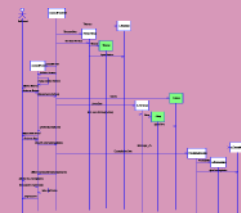
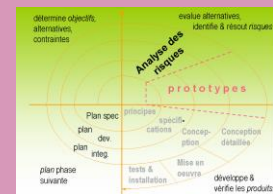
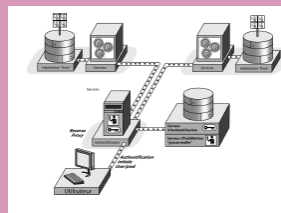
Création de bases de données

Accueil

Apprentissage

PAE

Evaluation



Localisation : U02-E03-S01

Sommaire

1.	Création de la base de données	3
1.1.	Choix du classement	3
1.2.	Le mode de récupération.....	4
1.3.	Le niveau de compatibilité	5
1.4.	Les fichiers de la base de données	6
	Création d'une base de données.....	9
1.5.	Répartition des données.....	11
2.	Création d'objets de base de données.....	13
2.1	Les types de données système	13
2.2	Les types de données utilisateur	15
2.3	Création de tables	15
2.4	Intégrité des données.....	18
2.5	Les contraintes.....	18
3.	Gestion des index	23
3.1	Introduction	23
3.2	Création des index	24
3.3	Utilisation de l'instruction CREATE INDEX.....	26

1. Création de la base de données

Les bases de données SQL Server sont stockées dans des fichiers.

Ces fichiers sont créés automatiquement lors de la création de la base.
La taille qui leur est allouée est celle spécifiée lors du processus de création.

Lorsque vous créez une nouvelle base de données, vous devez faire particulièrement attention aux options suivantes accessibles depuis les propriétés de la base de données.

Classement :	<input type="text" value="French_CI_AS"/>
Mode de récupération :	<input type="text" value="Complet"/>
Niveau de compatibilité :	<input type="text" value="SQL Server 2008 (100)"/>

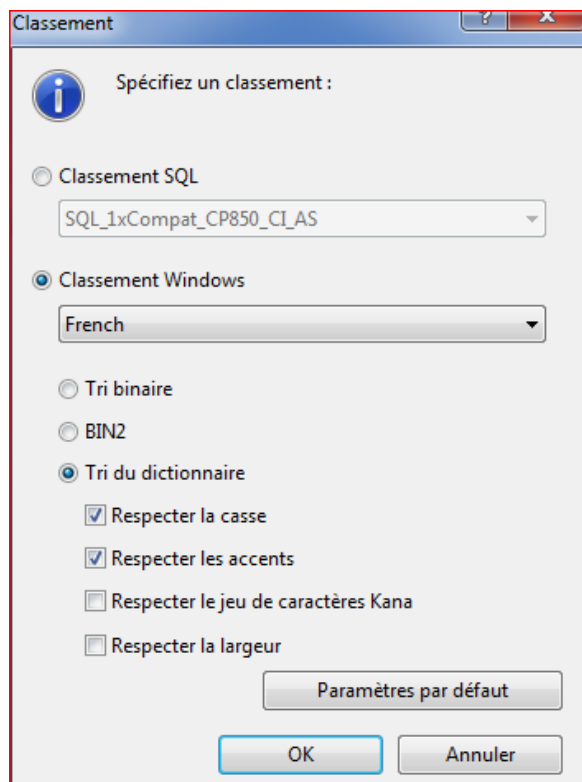
1.1. Choix du classement

Le classement détermine la manière dont les données textuelles sont ordonnées.
Dans le contexte d'une base de données en culture française, le classement est celui du dictionnaire français : Insensible à la casse CI mais sensible à l'accentuation AS.

Attention lorsque vous manipulez des données dans un contexte international avec des bases dont les valeurs des colonnes de type caractère et texte peuvent être classées différemment.

Le classement par défaut peut être modifié au niveau :

- Du serveur
- De la base de données à l'aide de la clause COLLATE de CREATE DATABASE
- D'une colonne à l'aide de la clause COLLATE associée à CREATE TABLE



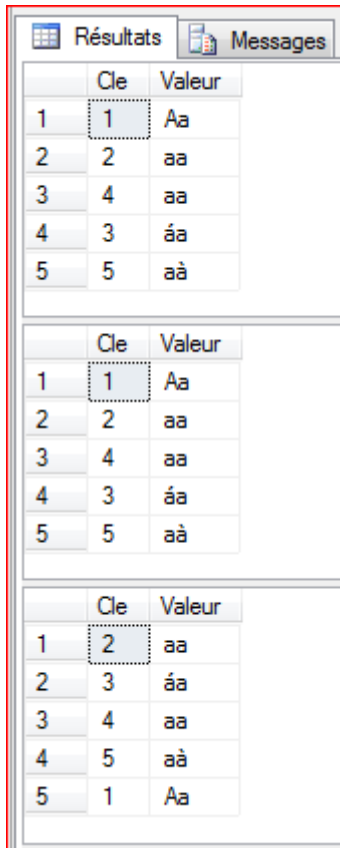
Vous pouvez changer le classement de la valeur d'une colonne texte en mode Design.

Cette fenêtre de dialogue est accessible depuis la propriété classement du concepteur de table.

Pour modifier le classement via le langage de description de données.

```
CREATE TABLE dbo.Table_1  
(  
    ColonneTri nchar(10) COLLATE French_CS_AS NULL  
) ON [PRIMARY]
```

Vous pouvez changer temporairement la manière dont sont ordonnées les valeurs en TRANSACT SQL lors de l'extraction des données (SELECT) comme le figure l'exemple ci-dessous.



	Cle	Valeur
1	1	Aa
2	2	aa
3	4	aa
4	3	áa
5	5	aà

	Cle	Valeur
1	1	Aa
2	2	aa
3	4	aa
4	3	áa
5	5	aà

	Cle	Valeur
1	2	aa
2	3	áa
3	4	aa
4	5	aà
5	1	Aa

```
CREATE TABLE #TestClassement (
    Cle int identity,
    Valeur nvarchar(100) NOT NULL
);
GO

INSERT INTO #TestClassement VALUES ('Aa');
INSERT INTO #TestClassement VALUES ('aa');
INSERT INTO #TestClassement VALUES ('áa');
INSERT INTO #TestClassement VALUES ('aa');
INSERT INTO #TestClassement VALUES ('aà');

SELECT Cle, Valeur FROM #TestClassement ORDER BY Valeur

SELECT Cle, Valeur FROM #TestClassement ORDER BY Valeur
        COLLATE French_CI_AS;

SELECT Cle, Valeur FROM #TestClassement ORDER BY Valeur
        COLLATE French_CS_AI;

GO

DROP TABLE #TestClassement;
```

1.2. Le mode de récupération

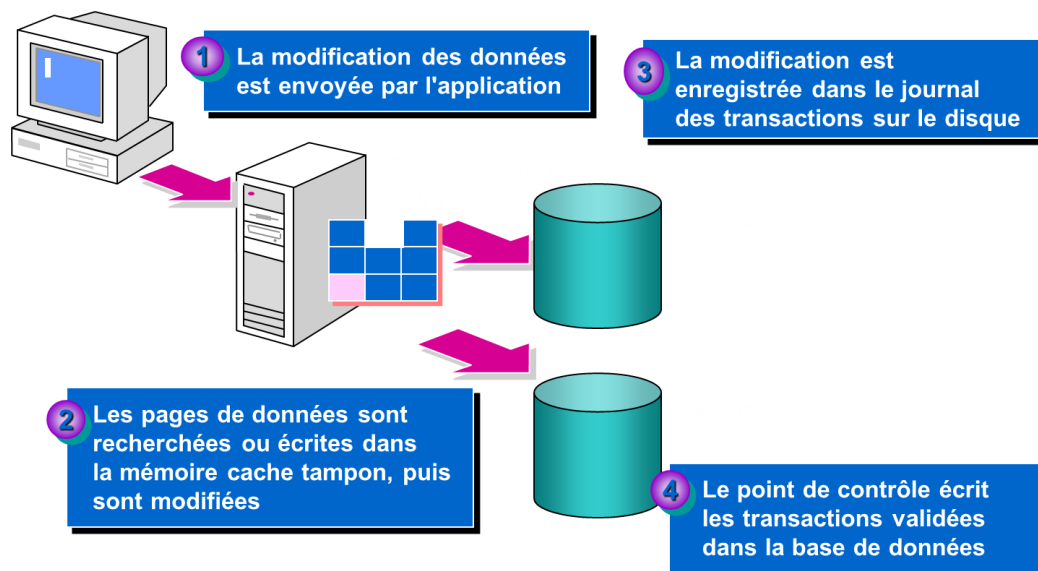
Il permet de préciser si les transactions doivent être conservées ou non après validation. Il est possible de mettre en place un serveur de secours en s'appuyant sur les journaux de transaction.

- Le mode de récupération **Complet** permet de s'appuyer sur le journal des transactions pour restaurer une base de données à un moment donné. Dans ce mode, les transactions validées sont conservées dans le fichier. Le fichier est épuré lors de sa sauvegarde.
- Dans le mode de récupération **Simple** seules les transactions en cours (avant commit ou rollback) sont présentes dans le fichier journal.

Fonctionnement du journal de transaction.

Le journal des transactions consigne les modifications apportées aux données, via les instructions INSERT, UPDAT ou DELET.

1. Une demande de modification est demandée.
2. Lorsque la demande de modification est exécutée, les pages de données concernées sont chargées dans une zone mémoire appelée cache de données
3. Chaque instruction de modification de données est enregistrée dans le journal des transactions au moment où elle est exécutée. La modification est toujours enregistrée dans le fichier journal, avant d'être appliquée à la base (journal à écriture anticipée).
4. Le processus de point de contrôle écrit toutes les transactions validées (par une instruction COMMIT TRANSACTION) dans la base. Si une erreur survient au cours d'une transaction, celle-ci est abandonnée et les données conservent leur état antérieur à la transaction opérations



En cas de défaillance du système, le processus de récupération automatique s'exécute au redémarrage de SQL Server.

1.3. Le niveau de compatibilité

Il est nécessaire d'être attentif au choix de cette option si la base de données en cours de création doit être déployée sur des serveurs s'exécutant dans une version antérieure de SQL Server.

Le mode **100** pour le niveau de compatibilité SQL Server SQL Server 2008.

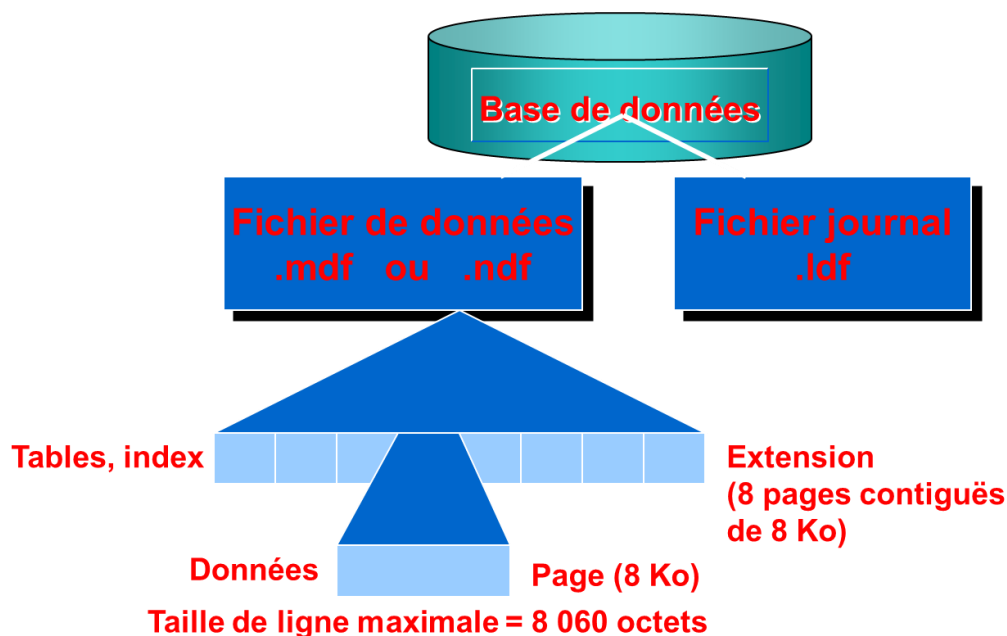
80 pour le niveau de compatibilité SQL Server 2000

90 pour un niveau de compatibilité SQL Server 2005

1.4. Les fichiers de la base de données

Il existe 3 types de fichier :

- **Primaire** (d'extension .MDF) : Obligatoire, il contient les objets systèmes et les objets utilisateur.
- **Secondaire** : facultatif (d'extension .NDF). Il contient toutes les données et objets qui ne se trouvent pas dans le fichier primaire. Lors du processus de création d'un objet, il sera alors nécessaire de préciser dans quel fichier l'objet devra être stocké.
- **Journal** : obligatoire. Il contient toutes les informations relatives aux transactions effectuées et permet de récupérer la base en cas d'incident, si la base est en mode de récupération complet.



Considérations relatives aux fichiers de base de données

La création d'une base de données s'effectue par copie de la base Model, comprenant les tables système et différentes options par défaut.

La taille occupée par une base de données est fonction :

- des choix retenus lors de la création, taille initiale
- du nombre de tables et d'index ainsi que des volumes de données
- du nombre de colonnes et de leur type

Les données sont stockées dans des pages de 8 Ko d'espace disque contigu. Les lignes ne peuvent pas s'étendre sur plusieurs pages (fractionnement).

Les données volumineuses qu'elles soient textuelles ou binaires (BLOB Binary Large Object) sont donc stockées sous une autre forme.

Les tables et les index sont stockés dans des extensions (une extension = 8 pages contiguës soit 64 Ko) qui peuvent être partagés dans le cas d'objets peu volumineux. Lorsqu'une table dépasse 8 pages, elle utilise ses propres extensions (extension uniforme).

Création de base de données

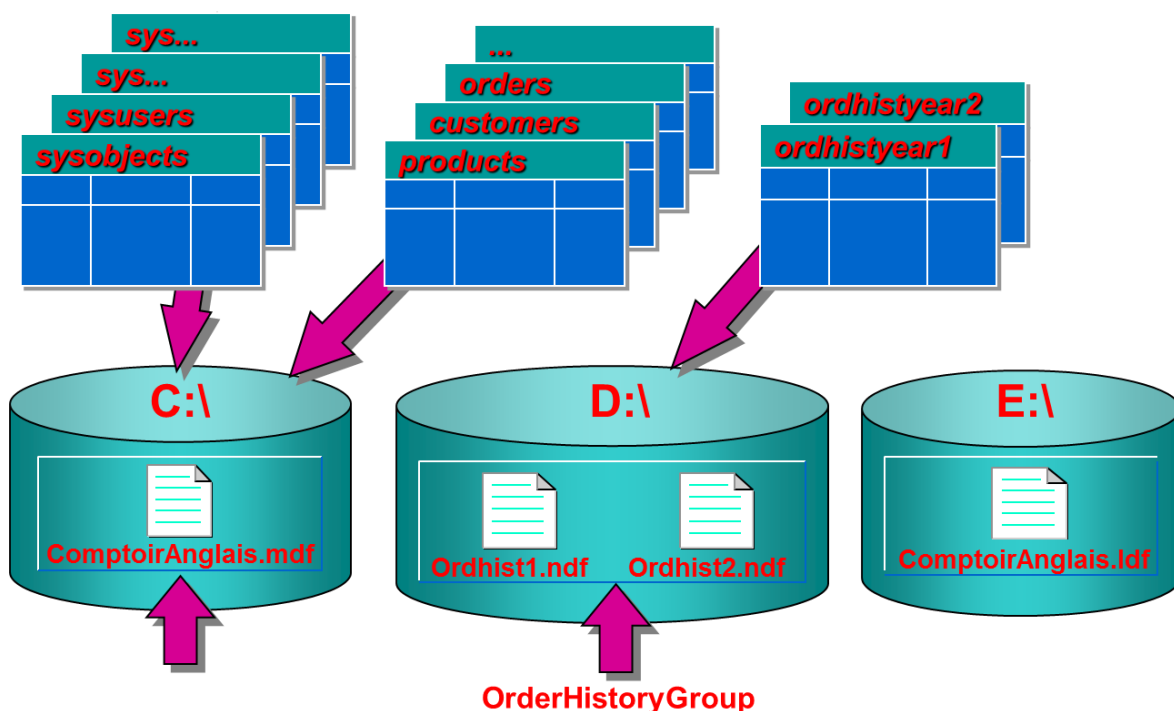
Lorsque vous créez la base de données, vous pouvez préciser la taille initiale de votre base de données et définir ses facteurs d'accroissement. Les options de croissance automatique des fichiers (par défaut) évitent les tâches administratives liées à la croissance manuelle de la base ; elles réduisent également le risque que la base de données manque d'espace disque de façon inattendue.

Nom de la base de données : BaseBost					
Propriétaire : <par défaut>					
<input checked="" type="checkbox"/> Utiliser l'indexation de texte intégral					
Fichiers de la base de données :					
Nom logique	Type de fichier	Groupe de fich...	Taille initiale (Mo)	Croissance automatique	Chemin d'accès
BaseBost	Rows Data	PRIMARY	3	Par 1 Mo, croissance illimitée	E:\MSSQLR2\MSSQL10_50\MSSQLR2\MSSQL\DATA
BaseBost_log	Journal	Non applicable	1	Par 10 pour cent, croissance illimitée	E:\MSSQLR2\MSSQL10_50\MSSQLR2\MSSQL\DATA

Il est possible de répartir les objets tables dans des fichiers de données secondaires afin d'optimiser les mécanismes d'accès aux données et les méthodes de sauvegarde de celles-ci.

Dans l'exemple illustré ci-dessous, les données d'historique de commandes sont conservées dans un fichier par année. Ces fichiers d'historique sont regroupés dans un groupe de fichiers et sont hébergés sur un disque ne disposant pas de mécanismes de tolérance de pannes. Ce groupe de fichiers sera sauvegardé une fois par an.

Les données « vivantes » seront quant à elles stockées sur des disques performants qui tolèrent les pannes (voir système de redondance et de répartition RAIDx).



Un fichier primaire .mdf se trouve toujours sur le groupe PRIMARY, et contient les tables système. Il contient également tous les objets non affectés aux groupes de fichiers utilisateur.

Création de base de données

Quelques exemples des possibilités offertes lors de la création d'une base de données.

Exemple 1: Création d'une base de données appelée Ventes avec un fichier de données primaires de 1 Go et un fichier journal de 500Mo.

Le fichier de données primaire peut croître jusqu'à 5 Go par incrément de 5% ; le fichier journal peut croître par incrément de 25Mo.

```
CREATE DATABASE Ventes
ON PRIMARY
(NAME = Ventes_data,
FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventedata.mdf',
SIZE = 1GB,
MAXSIZE = 5GB,
FILEGROWTH = 5%)
LOG ON
(NAME = 'Ventes_log',
FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventelog.ldf',
SIZE = 500MB,
FILEGROWTH = 5%)
```

Exemple 2: On aurait pu créer cette base avec 2 fichiers de données

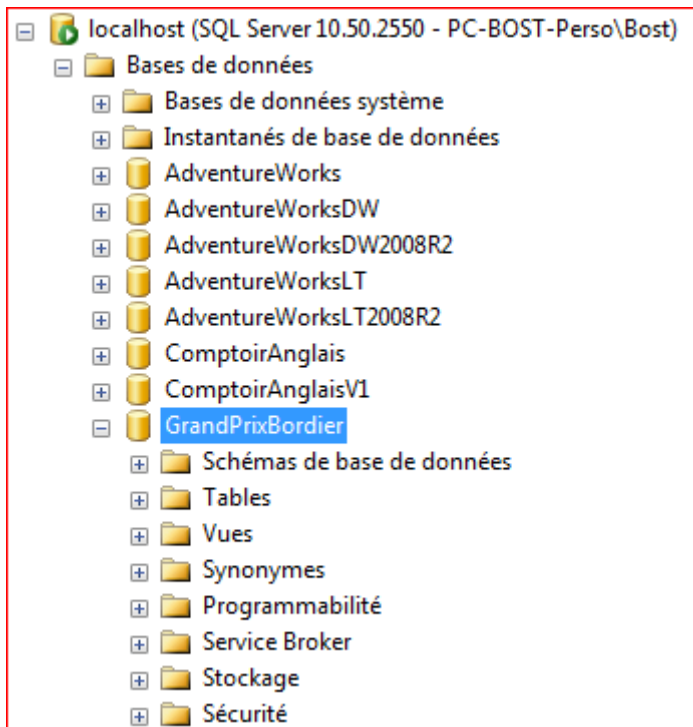
```
CREATE DATABASE Ventes
ON PRIMARY
(NAME = Ventes_data1,
FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventedata1.mdf',
SIZE = 1GB, MAXSIZE = 5GB, FILEGROWTH = 5),
(NAME = Ventes_data2,
FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventedata2.ndf',
SIZE = 500MB, MAXSIZE = 2GB, FILEGROWTH = 10%)
LOG ON
(NAME = 'Ventes_log',
FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventelog.ldf',
SIZE = 500MB, FILEGROWTH = 5%)
```

Exemple 3: On aurait pu créer cette base avec 2 fichiers de données, répartis dans 2 groupes de fichiers distincts

```
CREATE DATABASE Ventes
ON PRIMARY
(NAME = Ventes_data1,
FILENAME = 'D:\MSSQLSERVER\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\ventedata1.mdf',
SIZE = 1GB, MAXSIZE = 5GB, FILEGROWTH = 5),
FILEGROUP HistoVentes
(NAME = Ventes_data_2011,
FILENAME = 'E:\HistoSI\histoVentes2011.ndf',
SIZE = 250MB, MAXSIZE = 500MB, FILEGROWTH = 10%),
(NAME = Ventes_data_2012,
FILENAME = 'E:\HistoSI\histoVentes2012.ndf',
SIZE = 250MB, MAXSIZE = 500MB, FILEGROWTH = 10%)
LOG ON
(NAME = 'Ventes_log',
FILENAME = 'c:\program files\Microsoft SQL Server\mssql\data\ventelog.ldf',
SIZE = 500MB, FILEGROWTH = 5%)
```


Création d'une base de données

La vue Explorateur d'objets



Lorsqu'on développe le dossier base de données, ainsi que l'entrée d'une des bases, on affiche une liste de nœuds d'objets de la base :

- **Schémas de base de données** Schémas de la base ainsi que les informations en rapport
- **Tables**
Tables système (messagerie, plans de maintenance, réplication, sauvegarde/restauration, envoi des journaux) et utilisateurs
- **Vues**
Vues système et utilisateur
- **Synonymes**
Noms alternatifs pour des objets au niveau schéma ; les applications peuvent utiliser des synonymes pour accéder aux objets ; le nom de l'objet peut ainsi être modifié sans conséquence sur les applications
- **Programmabilité**
Représentation par nœuds de la plupart des types d'objets programmables (procédures stockées, déclencheurs, assemblys, types de données ...)
- **Service Broker**
Objets ayant un lien avec le service Broker : types de message, contrats, files d'attente ... Ce service permet de simplifier les traitements sur bases de données réparties.
- **Stockage**
Objets en rapport avec le stockage : catalogue de texte intégral ...
- **Sécurité**
Objets en rapport avec la sécurité : utilisateurs, rôles ...

Une base de données peut être créée à l'aide du concepteur de bases de données, en cliquant avec le bouton droit sur Base de données et dans le menu contextuel, en

sélectionnant Nouvelle Base de données ou grâce à l'instruction CREATE DATABASE de Transact-Sql.

Le journal des transactions sera créé automatiquement, quel que soit le procédé utilisé.

Dans la page **Général**, indiquer le nom et le propriétaire de la base ; une boîte de dialogue permet de sélectionner la connexion qui sera propriétaire de la base.

La création d'une base de données comprend l'affectation d'un nom, la déclaration de la taille et de l'emplacement des fichiers. ; on précise également la taille maximale autorisée et l'incrément de croissance autorisée (0 indiquera qu'aucune croissance n'est autorisée) :

Les groupes de fichier sont essentiellement destinés aux bases de données importantes (> 1GO) et aux tâches d'administration avancées : leur principal intérêt est d'améliorer le temps de réponse de la base, en autorisant la création de ses fichiers sur plusieurs disques et/ou l'accès par de multiples contrôleurs de disque.

Dans la zone chemin d'accès, saisir le chemin d'accès complet au fichier de données (emplacement par défaut sélectionné) ; le fichier primaire doit porter l'extension **.mdf**. Les fichiers de données secondaires constituent un espace supplémentaire pour stocker des données et doivent porter l'extension **.ndf**.

Exemple 4 : Création d'une capture instantanée SnapShot

Une capture instantanée de base de données étant une **vue en lecture seule d'une base existante**, un fichier journal ne peut pas être spécifié.

Conformément à la syntaxe, chaque fichier de la base de données source est spécifié et les groupes de fichiers ne sont pas spécifiés.

Les captures instantanées peuvent être utiles pour conserver une vue figée d'une base de données.

```
CREATE DATABASE Ventes_Snapshot ON
(NAME = Ventes_data1,
FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventedata1.mdf'),
(NAME = Ventes_data2,
FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventedata2.ndf')
AS SNAPSHOT OF VENTES
```

1.5. Répartition des données.

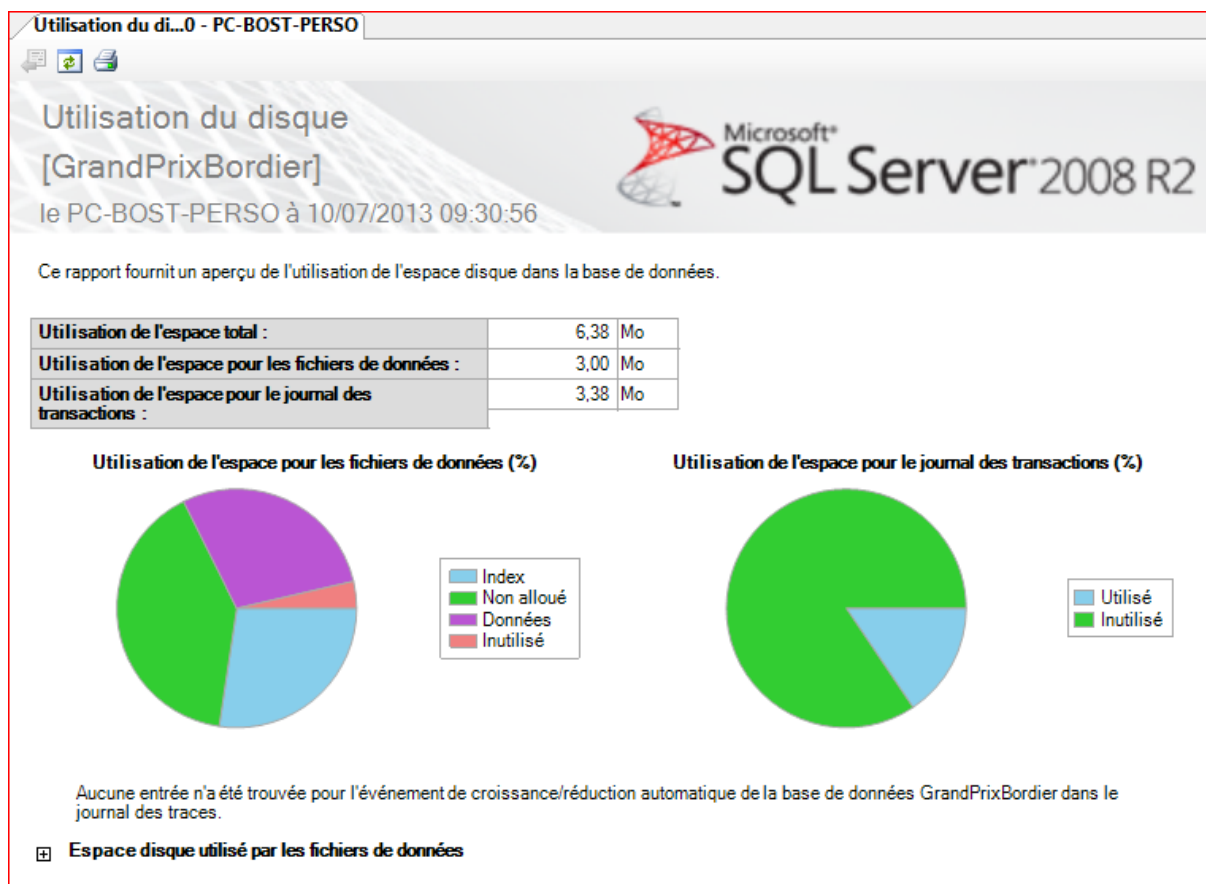
La répartition des données sur des fichiers ou des groupes de fichiers peut améliorer les performances (répartition des E/S), améliorer les temps de sauvegarde (sauvegarde par groupe de fichiers).

Le principal avantage des groupes de fichiers est de pouvoir répartir manuellement les objets sur un groupe de fichiers donné, alors que ce n'est pas possible sur un fichier de données.

Sous SQL Server Management Studio, vous créez une nouvelle base de données en sélectionnant dans le menu contextuel du noeud **Base de données**, l'option **Nouvelle Base de données**.

Le rapport Utilisation du disque fournit des informations sur l'utilisation de l'espace.

Répartition entre données et index



En développant, on obtiendra des informations sur l'espace disque utilisé par les fichiers de données. Ici, nous n'avons qu'un seul fichier de données.

☒ Espace disque utilisé par les fichiers de données

Nom du groupe de fichiers	Nom du fichier logique	Nom du fichier physique	Espace réservé	Espace utilisé
PRIMARY	AFPA_CDI2012	E:\MSSQLR2\MSSQL10_50\MSSQLR2\MSSQL\DATA\AFPA_CDI2012.mdf	4,00 Mo	3,25 Mo

Vous trouverez aussi d'autres rapports plus détaillés, comme celui présenté ci-dessous qui vous permet de connaître le nombre de lignes et l'espace réservé des tables et index.

Utilisation du disque par tables principales [GrandPrixBordier]					
le PC-BOST-PERSONA à 10/07/2013 09:32:04					
Ce rapport fournit des informations détaillées sur l'utilisation de l'espace disque par les 1 000 tables principales dans la base de données.					
Nom de la table	Nb. d'enregistrements	Réservé (Ko)	Données (Ko)	Index (Ko)	Inutilisé (Ko)
dbo.sysdiagrams	1	72	48	24	0
dbo.ResultatCourse	176	40	24	16	0
dbo.Soutenir_Pilote	7	16	8	8	0
dbo.Sponsor	22	16	8	8	0
dbo.Circuit	19	16	8	8	0
dbo.Ecurie	11	16	8	8	0
dbo.Pilote	22	16	8	8	0
dbo.PlanificationGP	19	16	8	8	0
dbo.GrandPrix	19	16	8	8	0
dbo.Fournisseur	4	16	8	8	0
dbo.Voiture	22	16	8	8	0
dbo.Soutenir_Ecurie	23	16	8	8	0
dbo.Affecter	22	16	8	8	0
dbo.Pays	236	16	8	8	0
dbo.HistoResultats	0	0	0	0	0

On peut également obtenir certaines informations par le biais du langage DBCC.

```
DBCC SQLPERF (logspace)
```

Affiche la taille et l'utilisation des fichiers journaux

Résultats Messages				
	Database Name	Log Size (MB)	Log Space Used (%)	Status
1	master	1,242188	41,19497	0
2	tempdb	0,4921875	56,05159	0
3	model	0,7421875	38,94737	0
4	msdb	5,554688	19,06646	0
5	ReportServer	6,242188	15,22997	0
6	ReportServerTempDB	0,8046875	56,18932	0
7	VolAvion	0,9921875	45,57087	0
8	ComptoirAnglais	4,117188	66,0816	0
9	AdventureWorksDW2008R2	1,992188	33,60294	0
10	AdventureWorksLT2008R2	1,992188	33,94608	0
11	AdventureWorks	1,992188	19,7549	0
12	AdventureWorksDW	1,992188	16,56863	0
13	AdventureWorksLT	1,992188	33,30882	0
14	GrandPrixLescure	3,367188	24,942	0

2. Création d'objets de base de données

2.1 Les types de données système

SQL Server propose plusieurs types de données système :

Type	Nb Octets	Plage
SQL Server		
Nombres entiers		
bigint	8	$(-2^{63} \text{ et } 2^{63} - 1)$
int	4	$(-2^{31} \text{ et } 2^{31} - 1)$
smallint	2	$-2^{15} (32768) \text{ à } 2^{15} - 1 (32767)$
tinyint	1	1 (0 à 255),
bit	1	0,1 ou NULL
Valeurs numériques exactes		
decimal[(p[,s])]	5 à 17	$(-10^{38} \text{ et } 10^{38} - 1)$
numeric[(p[,s])]	5 à 17	$(-10^{38} \text{ et } 10^{38} - 1)$
Valeurs numériques approximatives		
float[(n)] n<=24	4	-1,18E -38 à 3,40E +38
float[(n)] 24<n<=53	8	-1,79E + 308 à 3,40E +308
real	4	- 3,40E + 38 à 3,40E + 38
Valeurs binaires		
binary[(n)]	1 à 8000	
varbinary[(n)]		
varbinary[(max)]	jusqu'à $2^{31} - 1$ (+ pointeur de 2 octets)	
image	0 à $2^{31} - 1$ octets	
uniqueidentifier	16	
Nombres		
rowversion	8	Nombre unique
timestamp	8	(synonyme de rowversion)
Date et heure		
datetime	8(2 x 4 octets)	01/01/1753 au 31/12/999
smalldatetime	4(2 x 2 octets)	01/01/1960 au 06/06/2079
Monétaire		
money	8	
smallmoney	4	
Chaînes de caractères (1 octet par caractère)		
char[(n)]	n= 1 à 8000 caractères	
varchar[(n)]	n= 1 à 8000	
varchar[(max)]	jusqu'à $2^{31} - 1$ (+ pointeur de 2 octets)	
text	0 à $2^{30} - 1$ caractères	
Chaînes de caractères Unicode (2 octets par caractère)		
nchar[(n)]	n= 1 à 4000 caractères	
nvarchar[(n)]	n= 1 à 4000	
nvarchar[(max)]	jusqu'à $2^{31} - 1$ (+ pointeur de 2 octets)	

Les caractères Unicode utilisent 2 octets par caractère et prennent en charge tous les caractères internationaux. Choisir des longueurs fixes_lorsque la taille des données est cohérente, variable sinon

Les types de données numériques exactes permettent de spécifier exactement l'échelle et le degré de précision à utiliser.

Dans le cas des types de données numériques approximatives, les données sont stockées aussi précisément que possible.

Types particuliers

Sqlvariant (taille variable) : permet à une seule colonne de stocker plusieurs types de données. Les données, pour être manipulées, doivent être converties. Des limitations existent dans les usages de ce type. Je pense que la prudence nécessite qu'on évite de recourir à son usage. Il n'est pas toujours complètement pris en charge par les fournisseurs de données (OLEDB, ODBC ou JDBC).

table (taille variable) : utilisé pour stocker temporairement un ensemble de résultats pour traitements

Xml (taille variable) : utilisé pour stocker des données XML. Des extensions du langage de manipulation de données permettent d'élaborer des requêtes sur des données XML. Query ,... et des méthodes de modifications de données Insert(DML XML), Delete(DML XML), ReplaceValueOf(DML XML).

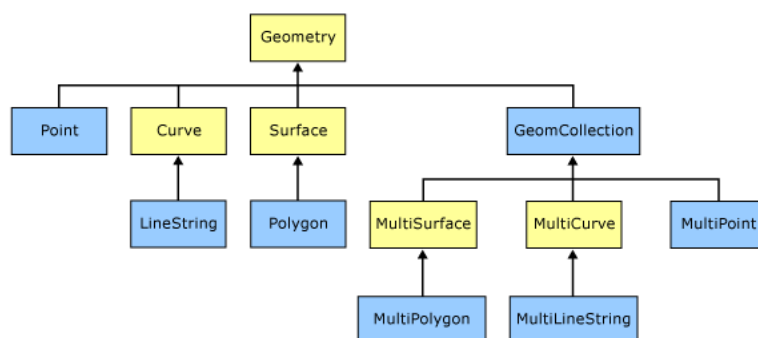
Json (taille variable) : utilisé pour stocker des objets avec la notation JavaScript. Des méthodes spécifiques existent pour extraire ou modifier les valeurs JSON_QUERY, JSON_VALUE,JSON_MODIFY

Les données spatiales.

Il existe deux types de données spatiales. Le type de données **geometry** prend en charge les données planaires, ou euclidiennes (monde en deux dimensions). Le type de données **geometry** se conforme à la fois à la spécification Open Geospatial Consortium (OGC) Simple Features for SQL version 1.1.0. et à la norme SQL MM (norme ISO).

De plus, SQL Server prend en charge le type de données **geography**, qui stocke des données ellipsoïdes (monde sphérique), telles que des coordonnées GPS de latitude et de longitude. Pour utilisateur averti... Il est toutefois nécessaire de s'y intéresser si vous souhaitez mettre en œuvre des fonctionnalités de géolocalisation et stocker des objets géométriques complexes tels que les coordonnées de pays, de routes, ...

La figure ci-dessous représente la hiérarchie **geometry** sur laquelle les types de données **geometry** et **geography** sont basés. Les types instanciables de **geometry** et **geography** sont indiqués en bleu.



2.2 Les types de données utilisateur

Ils permettent d'affiner les types de données afin d'assurer la cohérence entre les données communes à plusieurs tables. Il s'agit de types dérivés des types systèmes. Cette approche permet de spécialiser les types.

Un type de données utilisateur est déterminé pour une base spécifique.

L'instruction Transact-SQL CREATE TYPE permet de créer ces types de données.

Syntaxe sous Transact-SQL :

```
CREATE TYPE [nom_schema. ] type_name
{
    FROM type_système[(precision [, échelle ])]
    [NULL | NOT NULL ]] EXTERNAL NAME nom_assembly [ .nom_classe ]
}
```

Exemple 1 : création d'un type (code postal) de type caractère de 10 caractères n'autorisant pas les valeurs nulles

```
CREATE TYPE CodePostal FROM char(10) NOT NULL
```

L'approche de création de type CLR est trop éloignée de la norme. Aussi ne sera-t-elle pas abordée ici. Il s'agit du paramètre EXTERNAL NAME

2.3 Création de tables

Une table de base de données peut être créée à l'aide de SQL Server SQL Server Management Studio ou grâce à l'instruction CREATE TABLE de Transact-Sql.

SQL Server limite à:

- Deux milliards de tables par base de données
- 1024 colonnes par table
- 8064 octets par ligne (les types blob binary large object ne sont pas gérés de la sorte)

Dans chaque colonne, on spécifiera le nom et le type de données ainsi que, le cas échéant des contraintes et des valeurs par défaut (valeurs nulles autorisées ou non).

Une table sera modifiée par l'instruction Transact-Sql ALTER TABLE. ; Cette instruction permet d'ajouter, supprimer et modifier les colonnes.

Syntaxe sous Transact-SQL :

```
CREATE TABLE nom_table
( nom_colonne type_données [NULL | NOT NULL] [...n] )
```

Exemple 1

Création de base de données

Création de la table **Employe** pour laquelle les champs Matricule, Nom, Prénom, DateEntrée ne peuvent être nuls

```
CREATE TABLE Employe
(Matricule SMALLINT NOT NULL IDENTITY(1,1),
Nom VARCHAR(25) NOT NULL,
Prenom VARCHAR(15) NOT NULL,
Adresse VARCHAR(30) NULL,
Cp CodePostal,
DateEntree DATETIME NOT NULL,
DateFin DATETIME NULL)
```

La propriété **IDENTITY** permet de créer des colonnes contenant des valeurs numériques séquentielles générées par le système et identifiant de façon unique chaque ligne ajoutée à une table.

Une colonne d'identité sert le plus souvent la clé primaire.

Les paramètres précisent la valeur de départ et l'incrément ; il ne peut y avoir qu'une seule colonne d'identité par table ; une colonne d'identité doit être de type entier, numérique ou décimal et n'autorise pas les valeurs nulles.

Exemple 2 :

Création de la table Client dont la colonne **CodeClient** est créée avec le type de données uniqueidentifier, en utilisant une valeur par défaut générée par la fonction NEWID.

```
CREATE TABLE Client
(Codeclient UNIQUEIDENTIFIER NOT NULL DEFAULT NEWID( ),
NomClient VARCHAR(100) NOT NULL)
```

Résultat :

```
INSERT INTO [MEODB].[dbo].[Client]
([NomClient])
VALUES
('Bost')
```

Résultats		Messages
	Codeclient	NomClient
1	2A76CE76-C938-4657-99E4-B01E966C2E5E	Bost

Le type de données **uniqueidentifier** contient un numéro d'identification unique stocké comme une chaîne binaire sur 16 octets et sert à stocker un identificateur globalement unique (GUID).

La fonction NEWID crée un identificateur unique (GUID) stocké à l'aide du type de données uniqueidentifier.

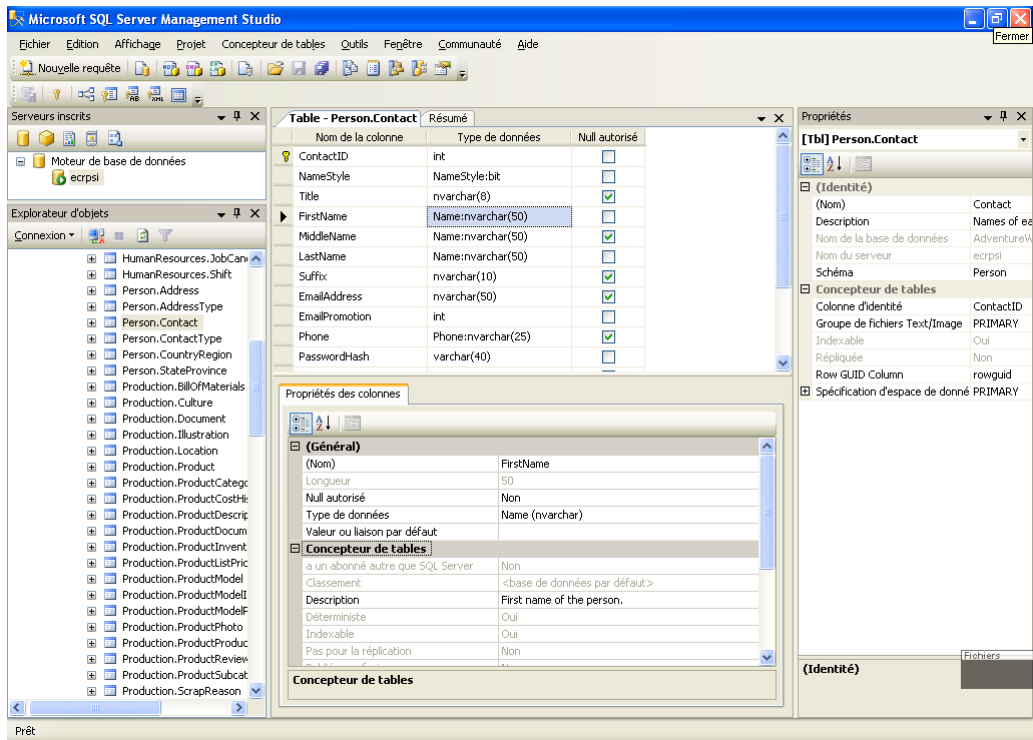
Le fait de laisser SQL Server fournir automatiquement des valeurs de clé plutôt que de laisser l'application cliente s'en charger peut améliorer les performances.

Sous **SQL Server Manager studio**, une nouvelle table sera créée en sélectionnant **Nouvelle table** dans le menu contextuel du nœud **Tables** de la base de données choisie.

Création de base de données

La table sera facilement conçue au moyen des vues fournies :

- la vue Table qui permet de visualiser l'ensemble des colonnes de la table
- la vue Propriétés des colonnes permet de configurer les paramètres de la colonne sélectionnée
- la vue Propriétés de la table permet de définir les propriétés générales de la table, son nom, sa description et son schéma ;



Partitionnement des tables

Les tables sont contenues dans une ou plusieurs partitions.

Partitionner les grandes tables permet de gérer des sous-ensembles de données, et peut améliorer les temps de réponse lors du travail sur les données de la table.

Par défaut, les tables ne comportent qu'une seule partition. Lorsqu'une table en contient plusieurs, les données sont partitionnées horizontalement, de sorte que des groupes de lignes sont mappés dans des partitions individuelles basées sur une colonne spécifique.

Exemple : Partitionner une table Commande suivant la date de commande ; la partition peut être alors divisée selon les plages de date, et chaque partition peut comporter des données annuelles, trimestrielles ou mensuelles.

Avant de créer une table partitionnée à l'aide de CREATE TABLE, il faut :

1. Créer une fonction de partition pour spécifier la manière dont la table est partitionnée à l'aide de l'instruction CREATE PARTITION FUNCTION.
2. Créer un schéma de partition pour spécifier les groupes de fichiers qui contiendront les partitions indiquées par la fonction de partition l'aide de l'instruction CREATE PARTITION SCHEME.

A tester...

2.4 Intégrité des données

On distingue 4 types d'intégrité :

- **L'intégrité de domaine** (ou de colonne) consiste à préciser l'ensemble des valeurs valides dans une colonne et à indiquer si les valeurs nulles sont autorisées.
- **L'intégrité d'entité** (ou de table) consiste à s'assurer que toutes les lignes d'une table possèdent un identificateur unique.
- **L'intégrité référentielle** garantit le maintien d'une relation existant entre une clé primaire dans une table référencée et une clé étrangère dans chacune des tables qui la référencent.
- **L'intégrité définie par l'utilisateur** permet de définir de règles d'entreprise spécifiques qui n'entrent dans aucune des autres catégories d'intégrité

2.5 Les contraintes

Les contraintes constituent une méthode pour assurer l'intégrité des données. Elles garantissent la validité des valeurs saisies dans les colonnes et le maintien des relations entre les tables.

Elles se mettent en œuvre dans les instructions CREATE TABLE ou ALTER TABLE

Les contraintes s'appliquent sur des colonnes ou sur des tables entières. Une contrainte de colonne fait partie de la définition de la colonne et ne s'applique qu'à celle-ci.

Une contrainte de table est déclarée indépendamment de la définition d'une colonne et peut s'appliquer à plusieurs colonnes d'une même table.

Dès lors que la contrainte porte sur plusieurs colonnes, elle doit être définie au niveau table.

Type de contrainte	Description	
Clé Primaire	PRIMARY KEY	Identifie chaque ligne de façon unique, interdit les doublons et garantit la création d'un index pour améliorer les performances ; les valeurs NULL ne sont pas acceptées. Une table ne peut comporter qu'une seule clé primaire
Unicité	UNIQUE	Empêche la création de doublons dans les colonnes non clé primaires et garantit la création d'un index pour améliorer les performances ; les valeurs NULL sont acceptées.
Clé étrangère	FOREIGN KEY	Définit une colonne ou ensemble de colonnes dont les valeurs sont égales à la clé primaire de la table référencée
Validation	CHECK	Spécifie une règle de validité s'appliquant aux valeurs d'une colonne.
Valeur par défaut	DEFAULT	Valeur par défaut de la colonne (lors d'un INSERT)
Champs non nuls	NOT NUL	La valeur de la colonne doit être définie

Exemple 1

Création de la table **Employe** pour laquelle :

- Les champs Matricule, Nom, Prénom, DateEntrée ne peuvent être nuls
- La clé primaire est le champ Matricule, champ compteur dont la valeur initiale est 1, et l'incrément de 1
- La colonne Nom ne peut contenir de doublons

```
CREATE TABLE Employe
( Matricule SMALLINT NOT NULL IDENTITY (1, 1) PRIMARY KEY,
  Nom VARCHAR (25) NOT NULL UNIQUE ,
  Prenom VARCHAR (15) NOT NULL,
  Adresse VARCHAR (30) NULL,
  Cp CodePostal,
  DateEntree DATETIME NOT NULL,
  DateFin DATETIME NULL)
```

Qu'on aurait pu coder :

```
CREATE TABLE Employe
( Matricule SMALLINT NOT NULL IDENTITY (1,1),
  Nom VARCHAR (25) NOT NULL ,
  Prenom VARCHAR (15) NOT NULL,
  Adresse VARCHAR (30) NULL,
  Cp CodePostal,
  DateEntree DATETIME NOT NULL,
  DateFin DATETIME NULL,
  CONSTRAINT PK_Employee PRIMARY KEY (Matricule),
  CONSTRAINT UK_Employee UNIQUE (Nom))
```

Création de base de données

Pour une clé primaire constituée de 2 colonnes, il est nécessaire de recourir à un objet Contrainte.

Il s'agit d'un cas d'école. Dans la réalité, la clé primaire ne serait pas constituée de la concaténation des valeurs nom et prénom.

```
CREATE TABLE Employee
(Nom          VARCHAR(25) NOT NULL ,
Prenom        VARCHAR (15)  NOT NULL,
Adresse       VARCHAR (30)   NULL,
Cp            CodePostal,
DateEntree    DATETIME      NOT NULL,
DateFin       DATETIME      NULL,
CONSTRAINT PK_Employee PRIMARY KEY (Nom, Prenom) ,
CONSTRAINT UK_Employee  UNIQUE (Nom) )|
```

En définissant une contrainte UNIQUE sur une ou plusieurs colonnes, SQL Server crée automatiquement un index unique (voir Index)

Exemple 2 : Contrainte DEFAULT

Ajout d'une contrainte DEFAULT qui insère la valeur '*Inconnu*' dans la colonne du prénom, si aucun prénom n'a été saisi.

```
ALTER TABLE Employee ADD
CONSTRAINT DF_Employee_Prenom DEFAULT '*Inconnu*' FOR Prenom
```

La colonne aurait pu directement être codée lors de la création de la table.

Cette contrainte ne s'applique qu'à l'insertion d'un enregistrement.

Exemple 3 : Contrainte CHECK

Ajout d'une contrainte CHECK qui assure que la date de résiliation du contrat est bien postérieure à la date d'embauche.

```
ALTER TABLE Employee ADD
CONSTRAINT CK_Date_Employee CHECK (DateFin > DateEntree)
|
```

Une contrainte **CHECK** peut être créée avec n'importe quelle expression logique (booléenne) qui retourne TRUE ou FALSE sur la base des opérateurs logiques.

Pour contrôler qu'une colonne salaire est comprise entre 1500€ et 10000€, l'expression logique est la suivante.

Salaire >= 1500 AND salaire <= 10000.

Pour s'assurer qu'une colonne Département est bien composée de 2 chiffres, l'expression logique sera de type

Dep like ('[0-9][0-9]')

Exemple 4 : Création de la table **Client** pour laquelle :

- Les champs CodeClient, NomClient, CodeReprésentant ne peuvent être nuls
- La clé primaire est le champ CodeClient
- La contrainte FOREIGN KEY, basé sur la colonne CodeReprésentant référence le champ CodeReprésentant de la table **Représentant** pour garantir que tout client est associé à un représentant existant

```
CREATE TABLE Client
(CodeClient SMALLINT NOT NULL IDENTITY(1,1),
NomClient VARCHAR(25) NOT NULL,
CodeReprésentant SMALLINT NOT NULL,
CONSTRAINT PK_Codeclient PRIMARY KEY (Codeclient),
CONSTRAINT FK_Client_Représentant FOREIGN KEY (CodeReprésentant)
REFERENCES Représentant (CodeReprésentant))
```

Pour pouvoir créer une telle relation au niveau du schéma de la base, il est nécessaire que la colonne CodeReprésentant soit clé primaire de la table REPRESENTANT.

Dans le cas d'une relation entre les tables Client et Représentant, le fait que l'intégrité empêche de supprimer un représentant est plutôt une bonne chose : un représentant quittant sa société, n'emmène pas tous ses clients avec lui

Entre les tables Commandes et Détail Commandes, nous pouvons mettre en place une suppression en cascade car les éléments de ligne commande n'ont pas d'existence en dehors de la commande.

Il faut également protéger l'insertion de données dans la table Détail Commandes, pour que toute ligne détail ait une correspondance dans la table Commandes.

Propriétés

[Rel] FK_Detail Commande_Commandes

(Général)

Spécification de tables et colonnes	
Colonnes clés étrangères	NO_Cde
Colonnes de clé Primary/Unique	NO_Cde
Table de base de clé étrangère	Détail Commande
Table de base de clé Primary/Unique	Commandes
Vérifier les données existantes à la création ou à	Oui
Concepteur de tables	
Appliquer la contrainte de clé étrangère	Oui
Appliquer la réplication	Non
Spécification INSERT et UPDATE	
Règle de mise à jour	Aucune action
Supprimer une règle	Cascade
Identité	
(Nom)	[FK_Detail Commande_Commandes]
Description	

Le choix effectué dans les spécifications INSERT et UPDATE (Supprimer une règle) dans les propriétés de la relation entre ces 2 tables garantissent les règles de gestion imposées.

```
ALTER TABLE.[Detail Commandes] ADD  
CONSTRAINT [FK_Detail Commandes_Commandes] FOREIGN KEY (NO_Cde)  
REFERENCES [Commandes] (NO_Cde) |  
ON DELETE CASCADE
```

La notion de mise à jour en cascade est à considérer avec prudence.

Les identifiants, clés primaires, sont par essence stables : leur valeur ne change jamais.

La mise à jour en cascade n'a donc pas vraiment d'intérêt.

Il est également possible d'opter pour d'autres implémentations.

Ainsi, il est possible de définir que toutes les valeurs qui composent la clé étrangère soient nulles ou définies à leur valeur par défaut si la ligne correspondante dans la table parente est supprimée.

Pour que cette contrainte soit exécutée, les colonnes de clé étrangère doivent accepter les valeurs NULL ou des valeurs par défaut.

Si une colonne peut avoir une valeur NULL et qu'il n'existe aucun ensemble de valeurs par défaut explicite, NULL devient la valeur par défaut implicite de la colonne SET DEFAULT.

```
ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }  
ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }
```

En l'absence de spécification particulière, aucune action n'est déclenchée sur la mise à jour ou la suppression d'une ligne d'une table parente.

3. Gestion des index

3.1 Introduction

Les index permettent d'optimiser les temps de réalisation des requêtes d'extraction de données (SELECT) sur des colonnes qui font l'objet de jointures, de filtre (clause WHERE), de tri (clause ORDER BY) ou de regroupement (clause GROUP BY).

Lors de l'exécution d'une requête d'extraction de données, le moteur de base de données exécute une opération d'analyse de la table ou l'accès via un index :

- Lors d'une **analyse de table**, SQL Server parcourt les pages et balaye ainsi toutes les lignes de la table en extrayant celles qui satisfont à la requête. (adapté aux tables de petite dimension ou lorsqu'un pourcentage élevé de lignes est renvoyé).
- Lors d'un **accès indexé**, SQL Server parcourt l'index ; il ne lit et n'extrait que les lignes satisfaisant les critères de la requête. (adapté à l'accès à des lignes uniques ou lorsqu'un faible pourcentage de lignes est renvoyé).

Lorsqu'un index existe, l'optimiseur de requêtes détermine s'il est plus efficace d'analyser la table ou d'utiliser l'index.

Les avantages et inconvénients de la création d'index.

De manière générale, les index accélèrent les requêtes qui effectuent des jointures de tables, des filtres, des opérations de tri ou groupage

Les index créés sont actualisés à chaque modification de données. La mise à jour des index, opération appelée calcul des index, exige des ressources mémoire et du temps processeur.

Vous pouvez définir un **index ordonné en clusters** par table.

Les lignes de données de chaque page sont stockées dans l'ordre de la clé d'index.

Chaque page de données contient une référence à la page qui la précède et à la page qui la suit afin de créer une séquence de pages ordonnées (liste à double lien nommé chaînage) qui permet aisément l'insertion d'une nouvelle page.

Par défaut, les index créés sur la ou les colonnes constitutives de la clé primaire sont ordonnés en cluster.

Pour les tables **dépourvues d'index ordonné en clusters**, les lignes de données ne sont pas stockées dans un ordre particulier et les pages ne sont pas chaînées.

Les colonnes à indexer :

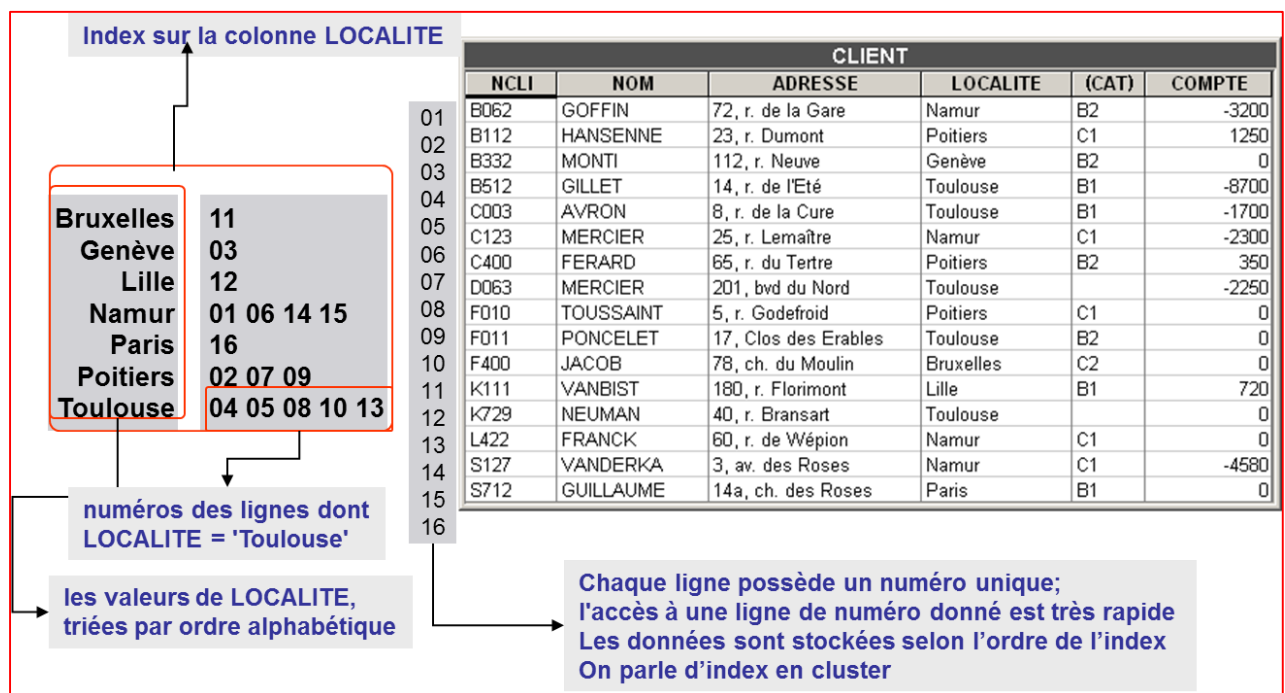
- Clés primaires, clés étrangères ou colonnes utilisées pour jointure
- Colonnes devant être triées (order by)
- Colonnes faisant l'objet de filtre d'égalité ou de type commence par
- Colonnes utilisées dans les fonctions d'agrégation

Les colonnes à ne pas indexer :

- Colonnes rarement référencées ou contenant des types bit, text ou image ou type CLR
- Colonnes contenant peu de valeurs uniques

Dans l'exemple qui suit, les lignes de données sont stockées selon un ordre défini par l'index en cluster sur la clé primaire ncli.

Un index est créé sur la colonne localité.



3.2 Création des index

L'instruction **CREATE INDEX** ou l'option **Nouvel Index** dans le menu contextuel du nœud **Index** de la table choisie de **SQL Server Management Studio** peuvent être utilisés pour la création d'index.

L'instruction **DROP INDEX** supprimera l'index.

Un index est automatiquement créé lorsqu'une contrainte **PRIMARY KEY** ou **UNIQUE** est ajoutée à une table.

Il est possible de créer des index sur des vues et des colonnes calculées. SQL Server stocke les informations sur les index dans la vue sys.indexes.

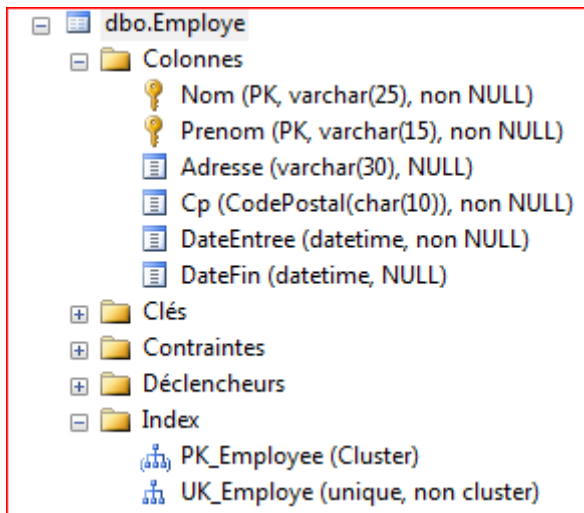
Il est possible d'indiquer qu'un index n'accepte pas les doublons. Il est toutefois préférable de lui substituer l'implémentation d'une contrainte **UNIQUE** plutôt qu'un index unique. L'instruction **CREATE INDEX** échouera s'il existe des doublons dans la colonne à indexer. Quel que soit le choix effectué, un index sera créé sur la colonne sur laquelle repose la contrainte d'unicité.

Un index peut porter sur plusieurs colonnes.

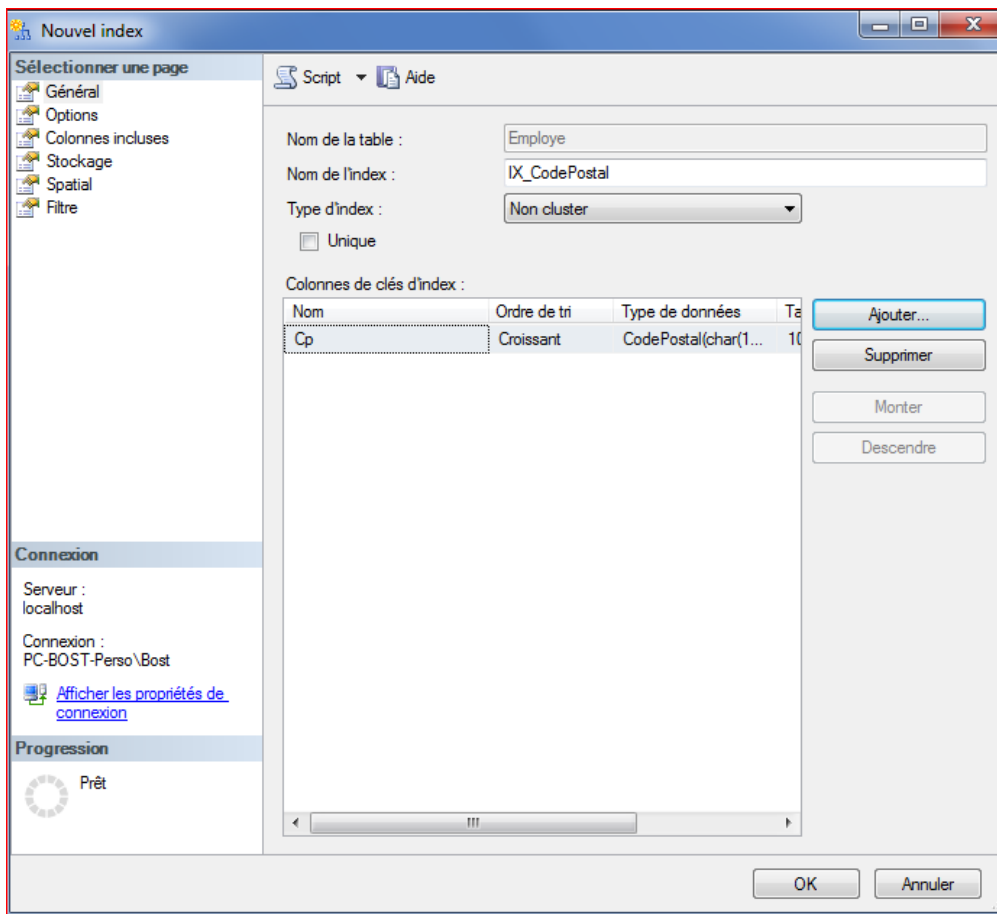
Création de base de données

La propriété FILLFACTOR permet de préciser le taux de remplissage d'un index. Cette propriété influe sur l'espace nécessaire au stockage des pages d'index et sur la périodicité des reconstructions de pages d'index lorsqu'une page d'index est pleine et doit être fragmentée.

Exemple des index créés par défaut sur la table Employe avec les contraintes de clé primaire et de colonne unique.



Création d'un index sur la colonne codePostal :



L'option **Propriétés** dans le menu contextuel d'un index donné du nœud **Index** de la table choisie de **SQL Server Management Studio** comporte plusieurs pages pour afficher et gérer des propriétés de l'index.

- La page Général affiche en particulier le nom, le type d'index et donne la possibilité d'ajouter ou supprimer des colonnes à l'index.
- La page Options permet de définir les options pour reconstruire l'index, recalculer les statistiques, définir le taux de remplissage, utiliser les verrouillages de page ou de ligne.
- La page Colonnes Incluses permet d'afficher et de gérer des colonnes supplémentaires aux colonnes d'index.
- La page Stockage liste la configuration de stockage
- La page Fragmentation permet d'afficher les données de fragmentation de l'index pour déterminer le besoin de réorganisation.

La fragmentation des données se produit lorsque des lignes sont ajoutées à des pages pleines qui doivent alors être séparées. Ce phénomène peut nuire aux performances ; il est donc nécessaire de reconstruire périodiquement les index fragmentés.

L'assistant paramétrage d'index permet d'analyser une charge de travail et émet des recommandations concernant les index à rajouter aux tables touchées.

Les options FILLFACTOR/ PAD INDEX permettent d'optimiser les performances des instructions INSERT et UPDATE. Lorsqu'une page est remplie, SQL SERVER doit séparer la page de manière à faire de la place pour les nouvelles lignes ; l'option FILLFACTOR permet d'indiquer le pourcentage du taux de remplissage souhaité ; le fait de laisser de l'espace libre dans chaque page réduit le recours à la séparation de page, mais augmente la taille de l'index.

3.3 Utilisation de l'instruction CREATE INDEX

Seul le propriétaire de la table ou de la vue peut créer des index sur celle-ci, qu'il y ait ou non des données dans la table.

Syntaxe sous Transact-SQL :

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX nom_index
ON { table | view } ( column [ ASC | DESC ] [ ,...n ] )
[ WITH
[ PAD INDEX = ON | OFF ]
[ [,] FILLFACTOR = taux_remplissage ]
[ [,] IGNORE_DUP_KEY = ON | OFF ]
[ [,] DROP_EXISTING ]
[ [,] STATISTICS_NORECOMPUTE ]
[ [,] SORT_IN_TEMPDB ]
]
[ ON groupe_fichiers ]
```

UNIQUE

Si le mot-clé UNIQUE n'est pas spécifié dans l'instruction CREATE INDEX, les doublons sont autorisés dans l'index.

CLUSTERED | NONCLUSTERED

Spécifie si l'ordre physique des lignes est dépendant / indépendant de l'ordre des clés : si le mot clé CLUSTERED n'est pas spécifié, c'est un index non ordonné en clusters qui est créé

PAD INDEX= ON | OFF

Spécifie si le taux de remplissage indiqué dans le paramètre FILLFACTOR est appliqué ou non aux pages d'index. L'option PAD_INDEX est utile seulement si FILLFACTOR est spécifié.

FILLFACTOR = *taux_remplissage*

Taux de remplissage des pages d'index (entre 1 et 100) ; si la valeur est 0 ou 100, les pages sont remplies complètement.

IGNORE_DUP_KEY= ON | OFF

Lors de l'insertion d'une clé en double, avec la valeur ON, seules les lignes qui ne respectent pas l'unicité sont omises ; avec la valeur OFF, l'ensemble de la transaction est annulée.

Exemple 1 :

Création d'un index sur le code postal

```
CREATE NONCLUSTERED INDEX [IX_CodePostal] ON [dbo].[Employee]
(
    [Cp] ASC
) ON [PRIMARY]
```

L'exécution de CREATE INDEX avec l'option DROP_EXISTING permet de reconstruire un index existant ou modifier ses caractéristiques :

- son type : possibilité de transformer un index non ordonné en clusters en index ordonné en clusters (l'inverse n'étant pas possible) ou transformation d'un index unique en index non unique et inversement.
- Ses colonnes indexées : possibilité d'ajouter ou retirer ou spécifier d'autres colonnes
- Ses options : modification possible des options FILLFACTOR/PAD_INDEX