



Dotnet France  
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

# Création et Gestion des tables

*Version 1.0*



Grégory CASANOVA

# Sommaire

---

1	Introduction.....	3
2	Pré-requis .....	4
3	Les tables.....	5
3.1	Les types de données .....	5
3.1.1	Les types de données Sql Server .....	5
3.1.2	Création d'un type de données par l'utilisateur .....	6
3.2	Créer une table.....	9
3.2.1	Avec du code T-SQL.....	9
3.2.2	Avec SSMS .....	10
3.3	Les contraintes d'intégrités .....	11
3.3.1	IDENTITY .....	11
3.3.2	PRIMARY KEY .....	12
3.3.3	UNIQUE.....	14
3.3.4	REFERENCE .....	15
3.3.5	DEFAULT .....	16
3.3.6	CHECK .....	16
3.4	Supprimer une table.....	17
3.4.1	Avec SSMS .....	17
3.4.2	Avec du code T-SQL.....	17
4	Manipulation de données dans une table.....	19
4.1	Ajout .....	19
4.1.1	Avec SSMS .....	19
4.1.2	Avec du code T-SQL.....	20
4.2	Modification .....	21
4.2.1	Avec SSMS .....	21
4.2.2	Avec du code T-SQL.....	21
4.3	Retrait.....	22
4.3.1	Avec SSMS .....	22
4.3.2	Avec du code T-SQL.....	22
5	Conclusion .....	24

## 1 Introduction

Les données ne peuvent pas être stockées directement dans la base. C'est pourquoi une base de données contient des sous ensembles, qui permettent de données une intégrité aux données qu'elle stocke. Ces sous ensembles sont appelés les tables. Les tables sont reconnues dans SQL Server par leur nom. Ces même sous ensemble, les tables, contiennent des sous ensembles, qui sont les colonnes, et qui sont les véritables conteneurs des données. C'est grâce aux colonnes que nous allons pouvoir stocker des données, et sauvegarder l'intégrité de nos données, en les rangeant, en différenciant leur types, ou encore en posant directement des contraintes sur ces colonnes. Enfin, une colonne est caractérisée par deux éléments : son nom et son typage, chose que nous allons voir dans la partie qui suivra les pré-requis.

## 2 Pré-requis

- Maitriser l'environnement de développement SQL Server à savoir SSMS (Chapitre 1).
- Savoir Gérer et configurer une base de données (Chapitre 2).

## 3 Les tables

### 3.1 Les types de données

#### 3.1.1 Les types de données Sql Server

L'une des forces de SQL Server, est sa diversité de types de données. Avec l'évolution du monde des entreprises et le besoin permanent de nouveaux types de données, dans la nouvelle version de SQL Server (2008), Microsoft a implémenté des nouveaux types de données telles que les données Géographiques, Géométriques, ou encore des nouveaux formats de données Time. Nous allons donc maintenant lister et expliquer tous les types de données contenus dans SQL Server 2008.

Char	Chaîne de caractères de longueur fixe d'un maximum de 8000 caractères.
Nchar	Chaîne de caractères Unicode, d'un maximum de 4000 caractères.
Varchar	Chaîne de caractères de longueur variable. Il est possible de préciser la valeur max, ce qui permet d'entrer des longueurs de chaînes de caractères de 2 <sup>31</sup> caractères.
Nvarchar	Chaîne de caractères Unicode, d'un maximum de 4000 caractères. En spécifiant max, le texte peut avoir une longueur maximum de 2 <sup>31</sup> caractères.
Int	Nombre entier compris entre -2 <sup>31</sup> et 2 <sup>31</sup> -1.
Bigint	Nombre entier compris entre -2 <sup>63</sup> et 2 <sup>63</sup> .
Smallint	Nombre entier compris entre -2 <sup>15</sup> et 2 <sup>15</sup> -1.
Tinyint	Nombre positif compris entre 0 et 255.
Decimal/Numeric	Nom exact de précision C (nombre entier) et D chiffres après la virgule tel que : Decimal (entre 0 et 38, 2) =Un nombre (2 chiffres après la virgule). Les valeurs supportées vont de -99999,999 à 99999,999.
Float	Nom approché de N chiffres tel que pour Float(N), N vas de 1 à 53.
Real	Identique à Float(24).
Money	Supporte les nombres monétaires compris entre -922337203685477,5808 et 922337203685477,5807 donc des nombres sur 8 octets.
Smallmoney	Supporte les nombres monétaires compris entre -214748,3648 et 214748,3647 donc des nombres sur 4 octets.
Date	Permet de stocker une donnée de type date comprise entre le 01/01/0001 et le 31/12/9999 avec la précision d'une journée.
Datetime	Permet de stocker une date et une heure sur 8 octets. Datetime a une précision accrue par rapport à Smalldatetime (précision de 3,33 millisecondes).
Datetime2	Il permet de stocker une donnée de type date et heure comprise entre le 01/01/0001 et le 31/12/9999 avec une précision de 100 nanosecondes.
Smalldatetime	Permet de stocker une date et une heure sur 4 octets. Les dates possibles vont du 1 <sup>er</sup> Janvier 1900 au 6 Juin 2079, avec une précision à la minute près.
Datetimeoffset	Il permet de stocker une donnée de type date et heure comprise entre le 01/01/0001 et le 31/12/9999 avec une précision de 100 nanosecondes. Les informations sont stockées au format UTC.
time	Permet de stocker des données positives inférieures à 24 heures. La précision peu être poussée jusqu'à 100 nano secondes.
Hierarchyid	Type de données propre à Sql Server qui va nous permettre de modéliser une structure hiérarchique dans une table relationnelle.
Geometry	Permet de travailler avec des données comprises dans un plan en 2 dimension, par exemple sur un trajet très court où on peu considérer que

	la terre est plate.
Geography	Ce type de données en revanche, ne nous permet pas d'assimiler la terre comme étant plane. On pourra l'utiliser pour travailler avec de grande distance. Il stocke la longitude et la latitude.
Uniqueidentif	Permet de créer un identificateur unique à l'aide de la fonction NEWID().
Binary	Supporte des données binaires sur n octets (De 1 à 255).
Varbinary	Supporte des données binaires sur n octets (De 1 à 8000). L'argument Max, permet de réserver 231 octet au maximum.
Bit	Valeur entière Booléenne pouvant prendre la valeur 0, 1 ou NULL.
Xml	Permet de stocker des documents Xml au sein d'une table dans une colonne.
Table	Type de données qui permet de renvoyer un ensemble de données en vue d'une utilisation future. Il est en particulier utilisé pour la création de tables temporaires.
Sqlvariant	Permet de stocker n'importe quel type de données hors mis les types text, ntext, timestamp et sql_variant. Sql_variant peut faire une taille maximale de 8016 octets.

Les types de données text, ntext et images existent toujours pour la compatibilité des anciennes versions, mais il est préférable d'utiliser les types varchar(max) et varbinary(max).

### 3.1.2 Création d'un type de données par l'utilisateur

Il est possible pour l'utilisateur, de créer ses propres types de données de deux manières, par SSMS ou avec la commande CREATE TYPE.

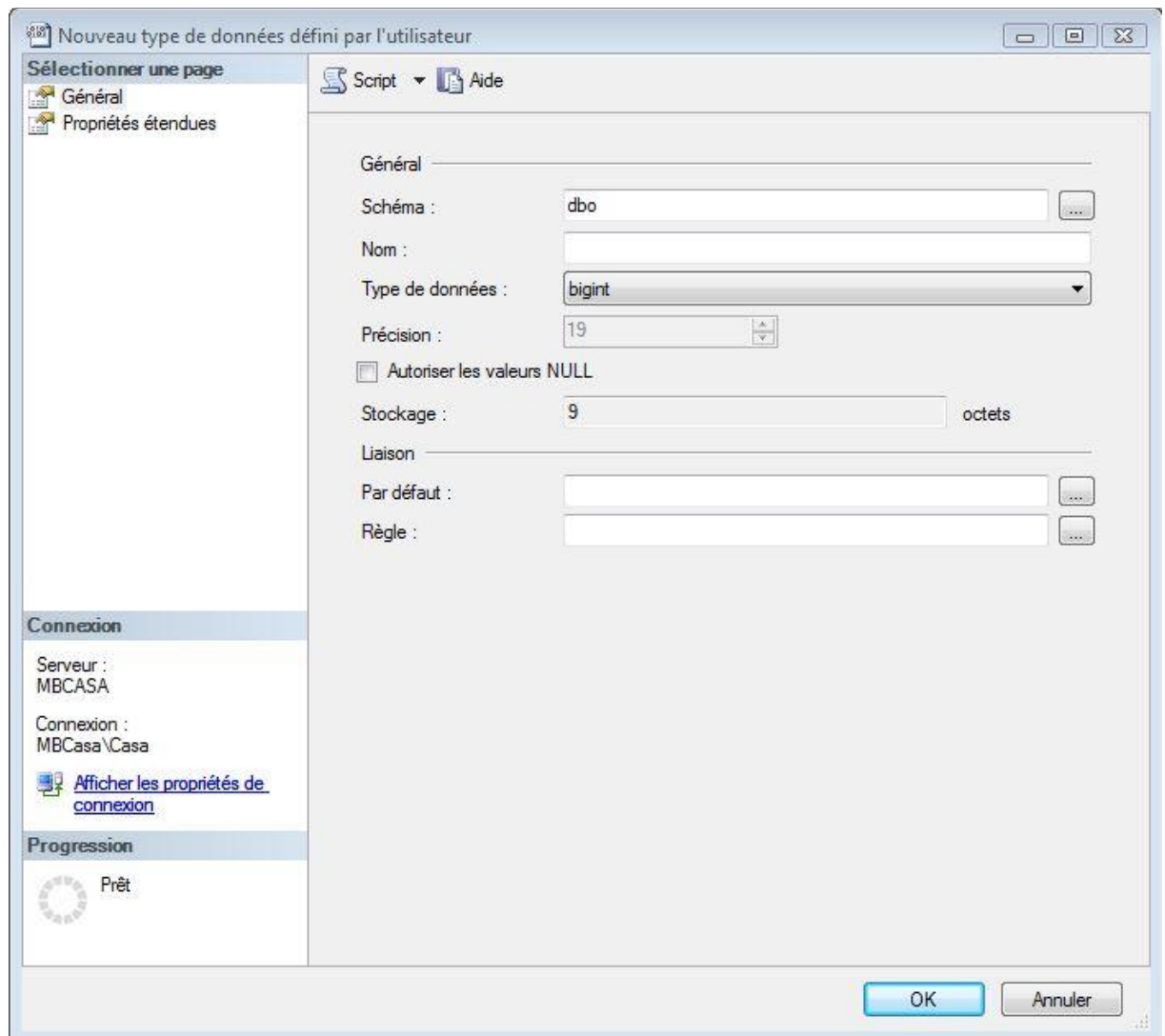
#### 3.1.2.1 Avec SSMS

Avec SQL Server Management Studio, il est possible de créer de nouveaux types de données personnalisés. Il vous suffit de déployer le nœud de votre base de données dans l'explorateur d'objet, puis le nœud programmabilité, et enfin le nœud Type. On remarque alors qu'il existe un sous nœud, qui se nomme « types de données définis par l'utilisateur ». Il suffira alors ensuite d'appliquer un click droit sur celui-ci et de sélectionner l'option « Nouveau type de données défini par l'utilisateur... » (Figure 1.1.2.1.1). Une nouvelle fenêtre apparaît (Figure 1.1.2.1.1). Il vous faudra renseigner des indications telles que le nom, le schéma ou encore le type de données supporté. Dès que vous aurez cliqué sur la touche de validation, votre nouveau type de données sera présent dans votre explorateur d'objet.

Pour supprimer ce type personnalisé, il vous suffit simplement de faire un click droit sur ce type dans l'explorateur d'objet, et de choisir supprimer.



Figure 1.1.2.1.1 :



### 3.1.2.2 Avec CREATE TYPE

Avec du code T-SQL, la forme générale de création de type personnalisé est la suivante.

```
CREATE TYPE Nom_Type
FROM Type_existant NULL
```

Analysons le code.

```
CREATE TYPE Nom_Type
```

La commande CREATE TYPE est bien entendu, la commande qui va nous permettre d'annoncer à Sql Server que nous allons créer un nouveau type de données. Il est donc nécessaire de renseigner à la suite le nom de ce nouveau type.

```
FROM Type_existant NULL
```

FROM annonce que nous allons utiliser un type existant pour définir notre propre type. Il est donc ensuite nécessaire de renseigner le type de données existant que nous utiliserons et si oui ou non, les valeurs NULL sont supportées.

Pour supprimer un type personnalisé de données, il suffit d'utiliser l'instruction DROP TYPE.

```
DROP TYPE NomSchema.Nom_Type
```

Après avoir exécuté le code, votre type de données sera supprimé.

## 3.2 Créer une table

### 3.2.1 Avec du code T-SQL

Comme pour la plupart des actions dans SSMS, l'ajout de tables peut se faire avec du code ou bien, avec l'interface graphique. Dans un premier temps, nous allons utiliser du code T-SQL. Nous verrons dans un prochain chapitre ce qu'est exactement du code T-SQL. Retenez pour le moment que c'est le langage de requêtage de base de données sous SQL Server.

```
CREATE TABLE Client
([Id_Client] int IDENTITY(1,1) PRIMARY KEY,
[Nom_Client] varchar(50) NOT NULL,
[Prenom_Client] varchar(50) NOT NULL,
[Nomero_Client] varchar(20) NOT NULL,
[Adresse_Client] varchar(50) NOT NULL,
[Mail_Client] varchar(50) UNIQUE NOT NULL);
GO
```

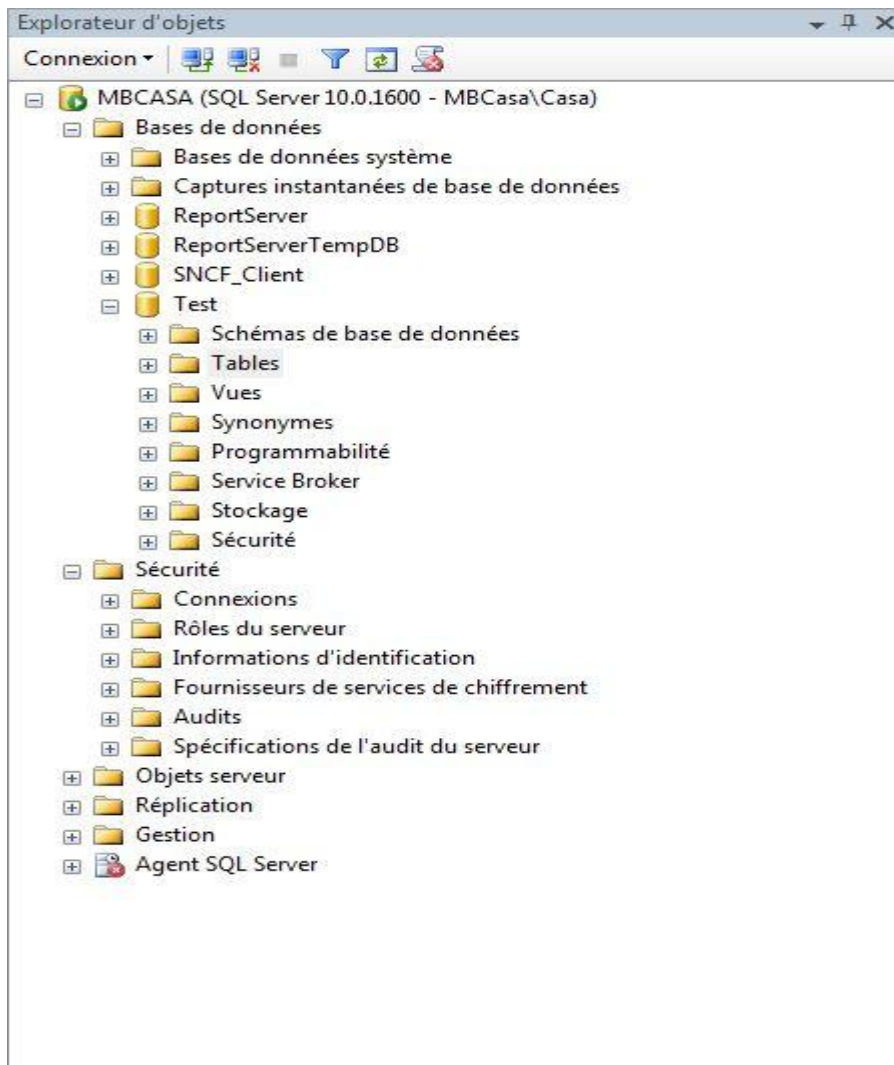
Décomposons ce code pour l'analyser.

```
CREATE TABLE Client
([Id_Client] int IDENTITY(1,1) PRIMARY KEY,
[Nom_Client] varchar(50) NOT NULL,
[Prenom_Client] varchar(50) NOT NULL,
[Nomero_Client] varchar(20) NOT NULL,
[Adresse_Client] varchar(50) NOT NULL,
[Mail_Client] varchar(50) UNIQUE NOT NULL);
```

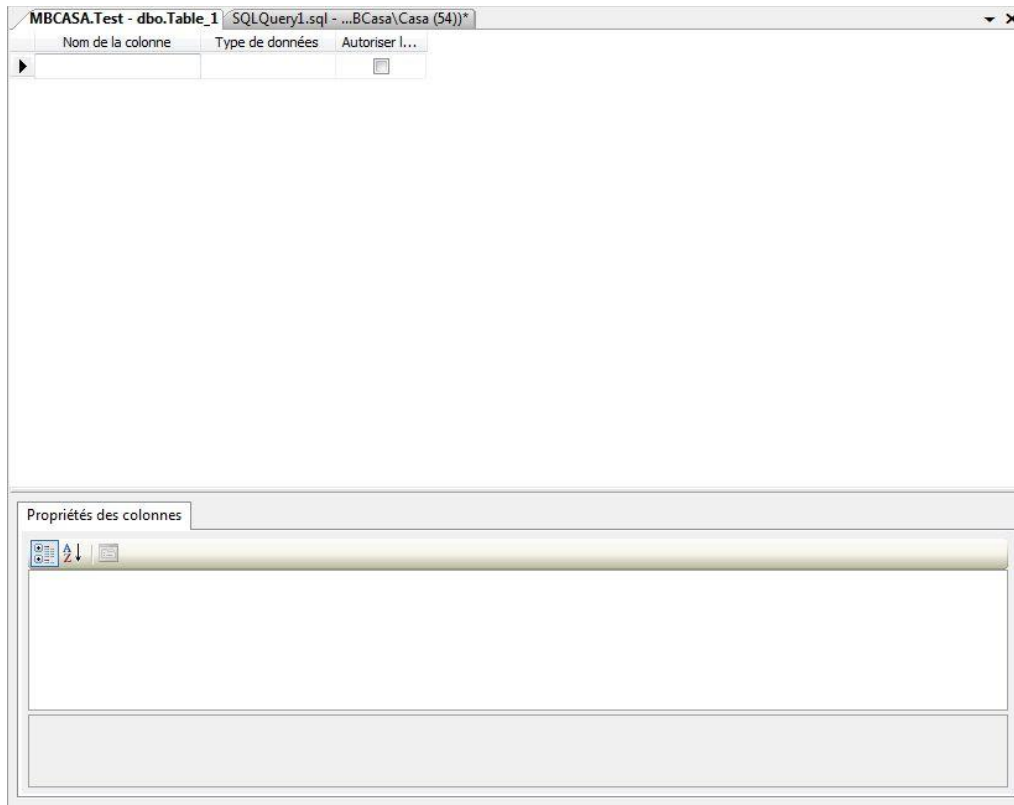
Le mot clé `CREATE TABLE` va bien entendu nous permettre de créer une table dans la base de données dans laquelle nous nous sommes rendus au préalable. Après ce mot clé, il est possible de spécifier le schéma de la table et le nom de la table séparé par un point. Nous verrons plus tard, dans la partie Administration de SQL Server, ce qu'est un schéma de base. Pour le moment, retenez qu'il est possible d'identifier un objet de la base de données par le modèle suivant : `NomBaseDeDonnées.NomSchéma.NomObjet`. Après cela, on placera entre parenthèses les colonnes que l'on veut créer, caractérisées par leur nom et le type de données qu'elles supportent. Il est aussi nécessaire de spécifier, si la colonne en question supporte ou non la valeur NULL. Lorsque vous aurez spécifié toutes ces caractéristiques, vous pouvez compiler votre code en appuyant sur F5 ou en cliquant sur la touche d'exécution du code. Vous aurez un message de validation de votre requête. Lors de la création de la table, il est possible de créer les contraintes d'intégrités, conformément à une base de données relationnelle qui sont les différents types de clés, mais aussi les contraintes telles que UNIQUE, IDENTITY... Nous expliquerons plus en détail par la suite, ce qu'est réellement une contrainte d'intégrité. Pour ajouter ces contraintes lors de la création de la table, il suffit de spécifier à la suite de la colonne voulue, le type de contrainte voulue, ainsi que son nom, et les arguments nécessaires à celle-ci.

### 3.2.2 Avec SSMS

Avec SQL Server management studio, la tâche est d'autant plus facile, puisque l'on utilise l'interface graphique. Dans votre explorateur d'objet, il vous suffit d'étendre le nœud de votre base de données, d'effectuer un click droit sur le nœud table et de sélectionner « Nouvelle table ».



Une nouvelle fenêtre apparaît alors comme ceci :



Vous aurez alors à renseigner les informations nécessaires, à savoir le nom de chaque colonne, avec le type de données qu'elle supportera et si elle supporte les valeurs NULL ou non. Pour sauvegarder votre table, il vous suffira d'effectuer un click droit sur l'onglet de la fenêtre où vous vous trouvez et de sélectionner « Sauvegarder ». Donnez alors un nom à votre table, et après rafraichissement de l'explorateur d'objet, vous pourrez voir votre table dans le nœud table de votre base de données.

### 3.3 Les contraintes d'intégrités

Comme dit précédemment, toute une gamme de contraintes existent pour assurer l'intégrité des données dans la base. Les contraintes s'appliquent exclusivement aux colonnes des tables et possèdent des caractéristiques propres.

#### 3.3.1 IDENTITY

Ce type de contrainte peu être affectée à une colonne par table, de type numérique entier. Elle permet d'incrémenter les valeurs d'une colonne, ligne après ligne. Par défaut, la contrainte IDENTITY part de 1, et a un pas d'incrément de 1. Il est possible de changer la valeur de départ et le pas d'incrément. Proposons un script qui créé une table, avec deux colonnes, une de type IDENTITY et une avec un type char, et faisons plusieurs insertions dans cette table.


```
--Cr  e la table avec la contrainte IDENTITY

CREATE TABLE MATABLE
(COLONNE1 NUMERIC(18,0) IDENTITY,
COLONNE2 char(10))

--Insertion multiple dans notre nouvelle table

INSERT INTO MATABLE
(COLONNE2)
VALUES
('Cours 1'), ('Cours 2'), ('Cours 3')
```

Remarquez que lorsque l'on ins re des lignes dans une table comportant une colonne IDENTITY, nous n'avons pas besoin de pr ciser la colonne et la valeur qu'elle prend en argument, d'o  son int r t, d'automatiser la saisie des donn es. V rifions maintenant le r sultat avec un simple SELECT :



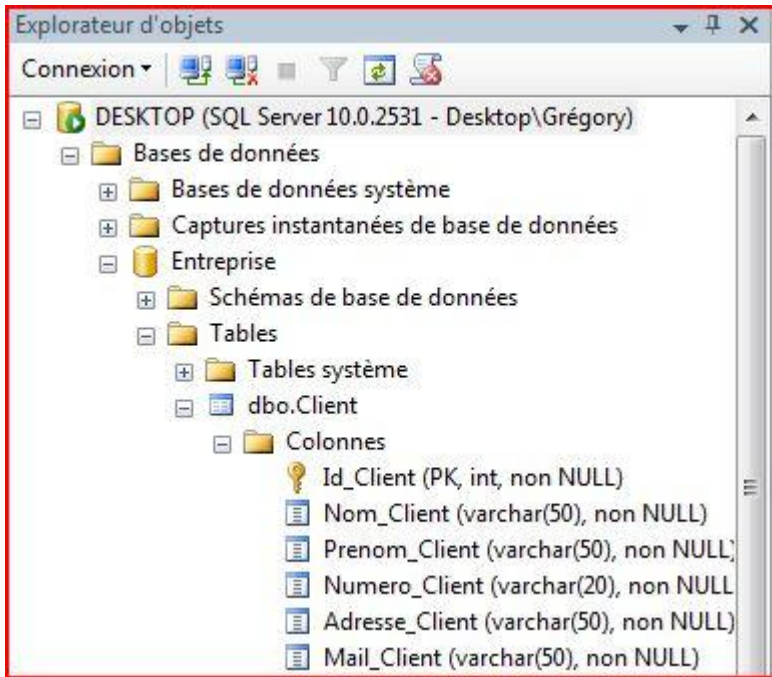
	COLONNE1	COLONNE2
1	1	Cours 1
2	2	Cours 2
3	3	Cours 3

On remarque bien que la colonne COLONNE1 c'est peupl e seule, gr ce   la contrainte IDENTITY. Il est bon de rappeler que nous n'avons droit qu'  une seule contrainte IDENTITY par table.

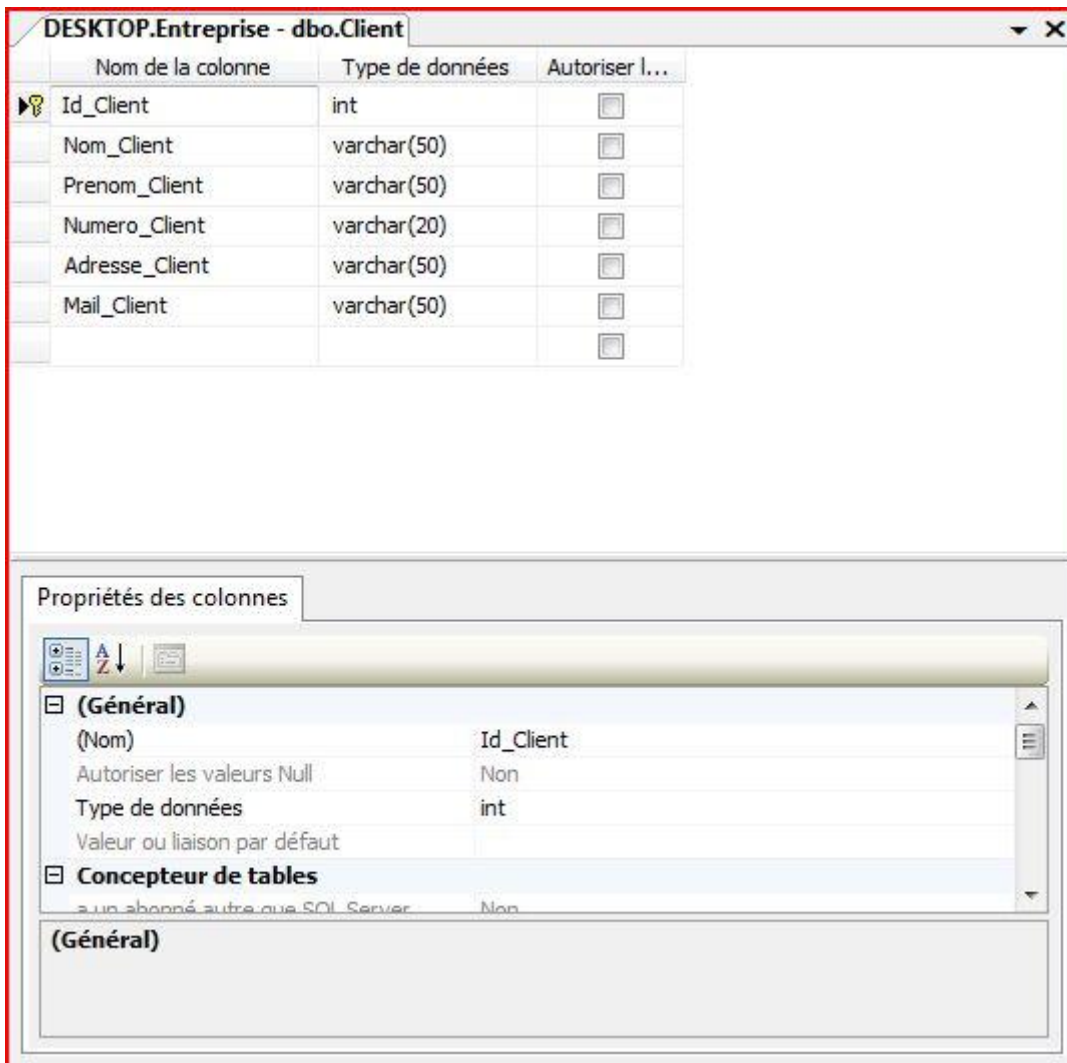
### 3.3.2 PRIMARY KEY

Cette contrainte permet de d finir une cl  primaire sur une ou plusieurs colonnes d'une table. Il ne peut y avoir qu'une seule cl  primaire par table, et la ou les colonnes sur lesquelles elle est d finie doivent  tre de type NOT NULL. Il est important de noter que lorsque nous cr ons une cl  primaire, un index unique est cr e (nous verrons cette notion plus en d tail plus tard dans les chapitres), on peut donc consid rer que les actions disponibles sur les indexs sont aussi disponibles lors de la cr ation d'une cl  primaire. Il y a trois fa on d'ajouter un cl  primaire : en la cr ant pendant la d finition de la table m me, en la cr ant apr s d finition de la table ou bien par SSMS. Commen ons par la fa on la plus simple, par SSMS.

-**Avec SSMS**, rien de plus simple qu' tablir une contrainte cl  primaire sur une colonne d'une table. Il vous suffit de d ployer dans l'explorateur d'objets tous les n uds qui m nent   votre table, comme ceci :



Effectuez un click droit sur la table en question et sélectionnez « création ». Cette nouvelle fenêtre apparaît à coté de votre explorateur d'objet :



Pour ajouter la clé primaire, faites un click droit sur la colonne voulue et sélectionnez « définir en tant que clé primaire ». Pensez ensuite à sauvegarder les changements de définition de table. L'opération est identique pour supprimer la clé primaire.

-**Pendant la définition de table**, il vous suffit d'ajouter le code suivant à la suite de la colonne voulue :

```
CREATE TABLE MATABLE1  
(COLONNE1 int CONSTRAINT PK_Nom_Contrainte PRIMARY KEY)
```

Le mot clé `CONSTRAINT` indique explicitement que nous allons définir une contrainte, à la suite de ce mot clé nous pouvons définir le nom unique de la contrainte. Cette définition n'est pas nécessaire, simplement, le nom est plus clair si on le définit soi-même. Si on ne le précise pas, celui-ci est généré automatiquement, et est composé d'une suite de caractères sans sens précis. Enfin le mot clé `PRIMARY KEY` indique que la contrainte est une clé primaire.

-**Après définition de la table**, admettons que nous avons créé une table avec le modèle suivant :

```
CREATE TABLE MATABLE1  
(COLONNE1 int)
```

Nous souhaitons alors ajouter une clé primaire sur la colonne COLONNE1. Cette opération est possible grâce à l'instruction DDL `ALTER TABLE` comme ceci :

```
ALTER TABLE MATABLE1  
ADD CONSTRAINT PK_PRIMARY  
PRIMARY KEY (COLONNE1)
```

Il est possible d'ajouter les options `CLUSTERED` ET `NON CLUSTERED`, suivant si nous voulons que l'index de la clé primaire généré automatiquement soit ordonné ou non.

### 3.3.3 UNIQUE

La contrainte `UNIQUE` comme son nom l'indique, va nous permettre de préciser sur une colonne, si les valeurs contenues dans celle-ci ne doivent pas être dupliquées dans plusieurs enregistrements. De ce fait, il ne sera pas possible avec une contrainte unique d'avoir deux fois une même valeur pour une colonne donnée. Enfin, contrairement à une table possédant une clé primaire, une table peut avoir plusieurs colonnes concernées par une contrainte `UNIQUE`. Lorsque une contrainte de ce type est définie, l'intégrité est gérée par un index de type `UNIQUE` créé en simultané. La définition d'une contrainte `UNIQUE` est simple avec du code T-SQL, puisque c'est de la même manière que nous avons créé notre clé primaire. Les deux façons (pendant ou après définition de la table) existent bien entendu :

```
--Pendant :  
  
CREATE TABLE MATABLE1  
(COLONNE1 int UNIQUE)  
  
--Après :  
  
CREATE TABLE MATABLE1  
(COLONNE1 int)  
  
ALTER TABLE MATABLE1  
ADD CONSTRAINT Nom_Contrainte UNIQUE
```

**Attention :** Il est possible d'ajouter une valeur NULL dans une colonne concernée par une contrainte unique et qui accepte les valeurs NULL, cependant, il n'est pas conseillé de pratiquer ceci.

**Rappel :** Lors de la création d'une contrainte `PRIMARY KEY`, la colonne est reconnue comme `UNIQUE` et n'accepte pas les valeurs NULL.

### 3.3.4 REFERENCE

La contrainte `REFERENCE` traduit la liaison qui existe entre une clé primaire et étrangère de deux tables. Il est conseillé de créer ce genre de contrainte qu'après la création de toutes les tables impliquées, sinon, lors de la compilation de votre script, des erreurs peuvent apparaître. Cette contrainte n'a pas de propriété particulière par défaut, il faut les ajouter soi-même, voyons dans un premier temps sa syntaxe. Nous présenterons les deux méthodes (pendant et après création des tables) par souci d'exhaustivité :

```
--Pendant :  
  
CREATE TABLE MATABLE1  
(COLONNE1 int PRIMARY KEY)  
  
CREATE TABLE MATABLE2  
(COLONNE1 int CONSTRAINT FOREIGN KEY COLONNE1  
REFERENCE MATABLE1 [COLONNE1]  
Options)  
  
--Après :  
  
ALTER TABLE MATABLE2  
ADD CONSTRAINT FOREIGN KEY COLONNE1  
REFERENCE MATABLE1 [COLONNE1]  
Options)
```

Comme dit précédemment, les deux choix sont possibles, seulement, lorsque les tables se multiplient dans votre script, il est de plus en plus difficile de gérer la création de ces contraintes pendant la création même des tables, puisque la table contenant la clé primaire doit exister avant celle contenant la clé étrangère. Il est d'ailleurs impossible de choisir cette option lorsqu'une table contient une clé primaire et une ou plusieurs clés étrangères. Le champ `Options` dans nos exemples constitue l'endroit même où nous pouvons définir les propriétés de nos références. Attardons nous sur ces propriétés possibles.

**NO OPTION :** C'est la valeur par défaut d'une contrainte `REFERENCE`. Elle permet d'obtenir le même comportement que si nous n'avions rien précisé.

**ON DELETE CASCADE** : Précise que si une ligne contenant la clé primaire référencée est supprimée, toutes les lignes contenant une clé étrangère référencée sur cette clé primaire seront supprimées, sous réserve que la clé primaire et la ou les clés étrangères possèdent la même valeur d'enregistrement.

**ON UPDATE CASCADE** : Permet de demander à SQL Server de mettre à jour toutes les clés étrangères référencées sur une clé primaire, lorsque cette clé primaire est mise à jour.

**SET NULL** : Lorsque la clé primaire référencée dans une table est supprimée, les clés étrangères de même valeur sont mises à NULL. Il faut ainsi faire attention à bien accepter les valeurs NULL pour la colonne spécifiée.

**SET DEFAULT** : Lorsque la ligne correspondant à la clé primaire référencée dans la table est supprimée, les valeurs pour la clé étrangère sont mises à la valeur par défaut définie sur la colonne en question.

### 3.3.5 DEFAULT

La contrainte **DEFAULT** est particulièrement utile pour éviter les valeurs NULL dans une table. Il faut toutefois garder à l'esprit qu'une valeur par défaut ne sera utilisée que dans le cas où l'utilisateur n'entre pas de valeur pour une colonne en particulier. Ce type de contrainte peut être appliquée pour toutes les colonnes mis à part les colonnes de type timestamp et celles qui possèdent une contrainte **IDENTITY**. Voici la syntaxe :

```
--Pendant :  
  
CREATE TABLE MATABLE1  
(COLONNE1 int DEFAULT Valeur)  
  
--Après :  
  
ALTER TABLE MATABLE2  
ADD CONSTRAINT DEFAULT Valeur  
FOR COLONNE1
```

**Remarque** : La définition des paramètres est toujours possible pendant ou après la création des tables concernées.

### 3.3.6 CHECK

Cette contrainte permet de vérifier, avant insertion ou mise à jour des données contenues dans la colonne en question, que les données à insérer sont bien au format voulu, ou encore qu'une valeur entrée dans la colonne pour un enregistrement appartiendra à un domaine de valeurs particulier. Regardons maintenant la syntaxe de cette contrainte :

```
--Pendant :  
  
CREATE TABLE MATABLE1  
(COLONNE1 int CHECK (expression_bouleenne))  
  
--Après :  
  
ALTER TABLE MATABLE2  
ADD CONSTRAINT CHECK (expression_bouleenne)
```

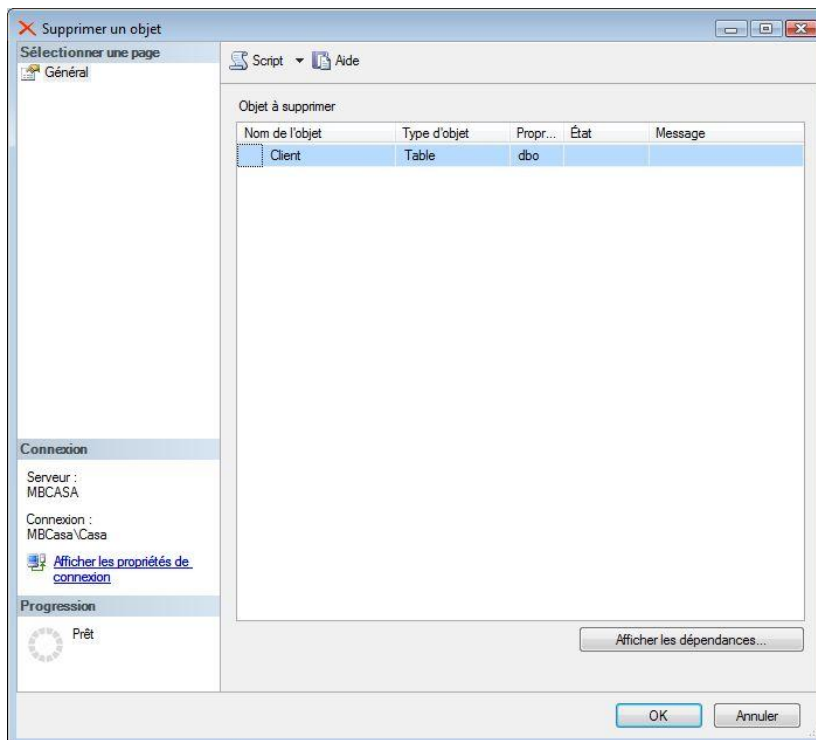
Il est possible d'ajouter l'option `NOT FOR REPLICATION` après le mot clé `CHECK`, afin de spécifier qu'il faut empêcher l'application de la contrainte dans un cas de réplication.

## 3.4 Supprimer une table

### 3.4.1 Avec SSMS

Avec SSMS, il est simple de supprimer une table de votre base de données. Il vous suffit de déployer votre base de données dans l'explorateur d'objet, d'effectuer un click droit sur la table choisie et de sélectionner « Supprimer ». Une nouvelle fenêtre s'affiche, il vous suffira de valider votre choix pour que l'action soit faite. (Figure 1.1.3.1)

Figure 1.3.1.1 :



### 3.4.2 Avec du code T-SQL

La structure de suppression de table avec du code T-SQL est la suivante.

```
USE Test
GO

DROP TABLE dbo.Client
GO
```

Analyse du code :

```
USE Test
GO
```

On indique que nous allons travailler dans la base de données Test.

```
DROP TABLE dbo.Client
GO
```

On précise alors que dans cette table Test, grâce à l'instruction DROP TABLE, nous allons supprimer la table Client dont le schéma est dbo. Après avoir exécuté ce code, on peut remarquer que la table que nous avons précisée après l'instruction DROP TABLE n'existe plus.

## 4 Manipulation de données dans une table

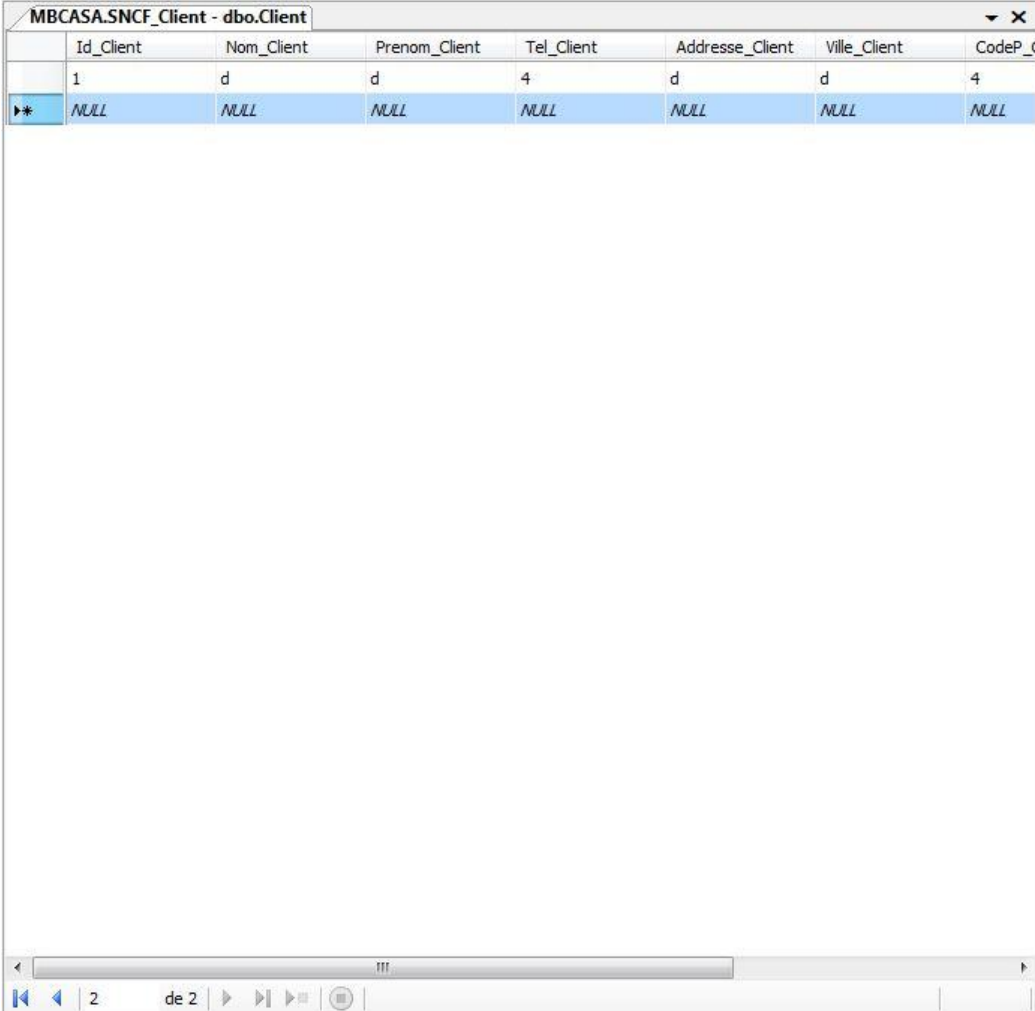
La manipulation des données et des objets dans SQL Server est toujours plus facile avec l'interface graphique. Cependant, avec de l'expérience, on peut trouver un avantage au fait de les manipuler avec du code T-SQL. C'est pour cela que nous allons présenter les deux manières, avec SSMS et avec du code T-SQL, pour ajouter, mettre à jour, ou bien modifier des données dans nos tables.

### 4.1 Ajout

#### 4.1.1 Avec SSMS

Avec SSMS, l'interface graphique nous permet encore une fois d'ajouter des données dans notre table très simplement. Il suffira juste de se rappeler des contraintes que nous nous sommes imposées lorsque nous avons choisis le type de données de chaque colonne. Pour ajouter des données dans chacune des cases d'une table de votre base de données, il vous suffira d'ouvrir l'Explorateur d'objet dans SSMS, de déployer votre base de données afin de rendre visible la table dans laquelle vous voulez ajouter des données, et enfin d'effectuer un clic droit sur cette même table et de sélectionner « Modifier les 200 premières lignes du haut ». Une nouvelle sous fenêtre s'affiche alors dans SSMS (figure 2.2.1.1), qui va vous permettre d'ajouter des données dans chacune des colonnes de votre table.

Figure 1.2.1.1



	Id_Client	Nom_Client	Prenom_Client	Tel_Client	Adresse_Client	Ville_Client	CodeP_C
	1	d	d	4	d	d	4
▶*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

### 4.1.2 Avec du code T-SQL

On pourra s’amuser à taper le code entièrement, mais il est nécessaire de rappeler qu’une autre des forces majeure de SQL Server est le gain de temps et de productivité. Il existe des structures de codes accessibles directement par l’interface graphique, qui permettent d’apprendre facilement lorsque l’on débute, ou bien d’aller plus vite dans un souci de productivité accrue. Pour générer ce code, il vous suffit de vous rendre sur votre table dans l’explorateur d’objet de SSMS, d’effectuer un click droit sur votre table, et de sélectionner l’option « **INSERT INTO** » dans le menu « Générer un script de la table en tant que ». Une nouvelle fenêtre de requêtes s’ouvre alors, et nous pouvons voir le code modèle généré par SSMS.

```
INSERT INTO dbo.Entrepos
(Nom_Entrepos)
VALUES
('Entrepos1')

GO
```

Détaillons ce code.

```
INSERT INTO dbo.Entrepos
```

La commande **INSERT INTO** est la commande d’insertion de données dans une table. Il est nécessaire d’ajouter la table de destination à la suite de cette commande de la façon suivante : Nom\_BDD.Schema\_Table.Nom\_Table.

```
(Nom_Entrepos)
```

Par la suite, il faut préciser les colonnes de destination des valeurs que nous allons entrer. Il est possible d’attribuer la valeur NULL à une colonne qui l’accepte de deux manières.

La première serait de ne pas ajouter le nom de la colonne dans la liste des colonnes de la table et de ne pas ajouter de valeur dans la liste des valeurs.

La seconde serait de laisser le nom de la colonne dans la liste des colonnes mais de préciser que la valeur est NULL.

```
VALUES
```

Le mot clé **VALUES** permet à SSMS d’identifier les lignes qui suivront comme étant des valeurs à insérer dans les colonnes et non pas comme des colonnes.

```
('Entrepos1')

GO
```

Il suffit alors ensuite de faire la liste des données à ajouter dans l’enregistrement. Il est important d’ajouter que pour des types de données caractères, il est nécessaire de mettre les données entre simple guillemets ( ‘ ’ ). Les données doivent être, comme présentées dans le modèle, données entre des parenthèses. Le mot clé **GO** la fin du lot.

## 4.2 Modification

### 4.2.1 Avec SSMS

La modification de données avec l'interface graphique se fait à la base de la même manière que l'ajout de données. Simplement, nous sommes confronté à une contrainte, puisque si nous avons plus de 200 enregistrements pour cette table, il ne sera pas possible de modifier la totalité de nos données, de plus, on ne pourra modifier les données qu'une par une contrairement qu'avec la méthode du code. Pour modifier des données dans chacune des cases d'une table de votre base de données, il vous suffira d'ouvrir l'Explorateur d'objet dans SSMS, de déployer votre base de données afin de rendre visible la table dans laquelle vous voulez modifier des données, et enfin d'effectuer un click droit sur cette même table et de sélectionner « Modifier les 200 premières lignes du haut ». Il vous est alors possible de modifier n'importe quelle valeur, de n'importe quelle ligne, de n'importe quelle colonne de votre table, du moment que la nouvelle valeur respectera les contraintes de type de données que vous aurez donné à cette colonne, ou encore que vous n'ayez pas mis une valeur NULL dans une colonne qui ne les supportent pas.

### 4.2.2 Avec du code T-SQL

Pour la modification des données avec du code, nous allons aussi opérer de la même manière que pour ajouter des données dans notre table. Nous allons générer ce code grâce à SSMS au lieu de le taper à la main. Pour cela, déroulez votre base de données dans l'explorateur d'objet. Effectuez un click droit sur la table dans laquelle vous voulez mettre à jour vos données, et sélectionnez UPDATE TO dans le menu, « Générer un script de la table en tant que ». Une nouvelle fenêtre de SSMS apparait alors, et elle vous présente le code de mise à jour de données dans une table donnée. Voici le code obtenu :

```
UPDATE [Entreprise].[dbo].[Client]
  SET [Nom_Client] = <Nom_Client, varchar(50),>
    , [Prenom_Client] = <Prenom_Client, varchar(50),>
    , [Numero_Client] = <Numero_Client, varchar(20),>
    , [Adresse_Client] = <Adresse_Client, varchar(50),>
    , [Mail_Client] = <Mail_Client, varchar(50),>
  WHERE <Conditions de recherche,,>
GO
```

Analysons-le en détail.

```
UPDATE [Entreprise].[dbo].[Client]
```

**UPDATE** est la commande de mise à jour de données de table. Il est nécessaire de spécifier la table de destination dans le format suivant : Nom\_BDD.Schema\_Table.Nom\_Table. Cette ligne de code indique que nous allons mettre à jour la table indiquée.

```
  SET [Nom_Client] = <Nom_Client, varchar(50),>
    , [Prenom_Client] = <Prenom_Client, varchar(50),>
    , [Numero_Client] = <Numero_Client, varchar(20),>
    , [Adresse_Client] = <Adresse_Client, varchar(50),>
    , [Mail_Client] = <Mail_Client, varchar(50),>
```

**SET** indique que nous allons spécifier en suivant, les colonnes dans lesquelles nous allons modifier les données. Vous pouvez de ce fait, ne choisir de mettre à jour que les colonnes que vous voulez.

```
WHERE <Conditions de recherche,,>  
GO
```

La commande `WHERE` indiquera les conditions dans lesquelles la mise à jour s'effectuera. On peut par exemple, choisir de modifier seulement les clients dont l'Id est égal à 6. L'instruction `GO` indique à SQL Server que le lot est terminé.

## 4.3 Retrait

### 4.3.1 Avec SSMS

Tout comme pour l'ajout et la modification de données, on peut supprimer des enregistrements grâce à l'interface graphique. La méthode est toujours la même, pour supprimer des données dans une table de votre base de données, il vous suffira d'ouvrir l'Explorateur d'objet dans SSMS, de déployer votre base de données afin de rendre visible la table dans laquelle vous voulez modifier des données, et enfin d'effectuer un click droit sur cette même table et de sélectionner « Modifier les 200 premières lignes du haut ». Une nouvelle fenêtre apparaît encore une fois, et vous pouvez choisir, grâce à un click droit, de supprimer une ligne de votre table.

### 4.3.2 Avec du code T-SQL

Nous allons encore une fois générer le code grâce à SSMS. Déroulez le nœud de votre base de données dans l'explorateur d'objet. Effectuez un click droit sur la table choisie, et sélectionnez l'option `DELETE TO` dans le menu « Générer un script de la table en tant que ». Le code suivant apparaît :

```
DELETE FROM [Entreprise].[dbo].[Client]  
WHERE <Conditions de recherche,,>  
GO
```

Analysons ce code :

```
DELETE FROM [Entreprise].[dbo].[Client]
```

L'instruction `DELETE FROM` indique que nous allons supprimer une ligne ou plusieurs lignes dans la table indiquée à la suite de cette instruction.

```
WHERE <Conditions de recherche,,>  
GO
```

La condition `WHERE` indiquera alors simplement dans quels cas on supprimera les lignes de notre table. On pourra par exemple supprimer tous les clients donc l'Id sera supérieur à 250. L'instruction `GO` indique que l'action est terminée.

Dans le cas où l'on voudrait supprimer toutes les données d'une table, il existe une commande bien plus rapide que `DELETE FROM`. Cette commande est la commande `TRUNCATE TABLE`.

```
TRUNCATE TABLE Schema.Nom_table
```

Avec la commande `TRUNCATE TABLE`, on indique à SQL Server qu'on va supprimer toutes les données contenues dans une table. On doit positionner en argument le schéma et le nom de la table. Après avoir exécuté ce code, votre table ne contiendra aucune donnée. L'avantage de `TRUNCATE`

`TABLE` est que son exécution est bien plus rapide que de supprimer les données avec simple `DELETE FROM`.

## 5 Conclusion

Dans ce chapitre, nous avons vu essentiellement comment utiliser au mieux les tables, comment les créer, les supprimer ou bien les mettre à jour. Nous avons aussi appris à mettre en place des contraintes sur des sous ensemble de ces tables, nommés des colonnes, afin de garder l'intégrité des données contenues dans nos tables et par extension, dans notre base de données. Les morceaux de code que nous avons pu exécuter ici font parti d'un langage nommé le T-SQL (Transact SQL), que nous allons développer dans le chapitre suivant.