



Collège
Lionel-Groulx

Département d'informatique

Cours : 420-KBA-LG, programmation de bases de données

Automne 2019

Programmation de bases de données

Transact-SQL (SQL Server)

Saliha Yacoub

COLLEGE LIONEL-GROULX

Table des matières

Historique des versions	5
Chapitre 1, pour bien commencer.....	6
Chapitre 2, installation, configuration et connexion.....	9
Mode d'authentification :	10
Authentification Windows.....	10
Authentification SQL server.....	10
Étape 1 : Changer le mode d'authentification	10
Étape 2 : Créer une nouvelle connexion	13
Étape 3 : Attribuer les rôles.....	15
Étape 4, Connexion avec l'authentification SQL Server et création de la base de données.....	17
Où est stockée la base de données ?	20
Chapitre 3, création des tables.....	24
Types de données SQL Server	24
La propriété « IDENTITY » d'une table	25
Création des tables avec SQL Server	28
Chapitre 4, le modèle de données avec SQL Server Management Studio.....	30
Étape 0 : création de la base de données	30
Étape 2 : Création des tables :.....	30
Étape 3, créer le schéma de la BD	31
Étape 4 : Définir les relations (la clé étrangère).....	33
Définir la clé primaire composée	35
Chapitre 5, éléments du langage Transct-SQL.....	37
Définitions	37
Éléments du langage Transact-SQL :	37
Les variables et leurs déclarations.....	37
Les mots réservés : BEGIN ...END	37
Les structures de contrôles	37
Les curseurs :.....	41
Chapitre 6, les procédures stockées.....	44
Définition	44
Avantages à utiliser les procédures stockées.....	44
Syntaxe simplifiée de définition d'une procédure stockée avec Transct-SQL	44

Exemple1 : Tous les paramètres sont en IN. (Insertion)	45
Exécution d'une procédure dans son SGBD natif (MS SQL Server)	45
Exemple 3, utilisation de LIKE dans une procédure stockée	46
Exemple 4 : Procédure avec un paramètre en OUTPUT.....	47
Les fonctions stockées : Syntaxe simplifiée.....	48
Cas d'une fonction qui ne retourne pas une table	48
Exemple 1, fonction avec paramètres.....	48
Exécution d'une fonction dans MS SQL Server	48
Exemple2 : fonction sans paramètres	49
Cas d'une fonction qui retourne une table.	49
Exemple	49
Supprimer une fonction ou une procédure :.....	50
En conclusion pour les procédures et les fonctions.....	50
Les procédures stockées et les fonctions : les Templates.....	52
Chapitre 7, les Triggers ou déclencheurs	54
Définition :.....	54
Rôle des triggers :.....	54
Syntaxe simplifiée :.....	54
Principe de fonctionnement pour les triggers DML	55
Exemple 1, suppression en cascade	55
Exemple 2	56
Exemple 3	56
RAISERROR:	57
Activer /désactiver un trigger.....	59
Supprimer un trigger.....	60
Retour sur la commande CREATE TABLE : ON DELETE CASCADE.....	60
En conclusion :.....	63
Chapitre 8, les transactions	64
Notions de Transactions :.....	64
Propriétés d'une transaction.....	64
Récupération d'une transaction.....	66
Récupération complète de la base de données	66
Transactions concurrentes	67

Perte de mise à jour	68
Les verrous	68
Chapitre 9, optimisation de requêtes.....	70
Introduction.....	70
Les index.....	70
Types d'index :.....	72
Les CLUSTERED INDEX :	72
Les index non CLUSTERED INDEX :	74
La commande CREATE INDEX.....	75
Suppression d'un index	76
Afficher les index définis sur une table	76
Outils de mesures des performances.....	76
Règles d'optimisation de requêtes :.....	76
Chapitre 10, introduction à la sécurité de données.....	77
Introduction.....	77
Menaces courantes :	77
Injection SQL.....	77
Élévation de privilège :	78
Détection des attaques et surveillance intelligente.....	79
Mots de passe.....	79
Rôles du serveur :	80
Rôles niveau bases de données :.....	81
Privilèges sur les objets (tables, colonnes, lignes) :.....	82
Par l'interface SQL Server Management Studio :.....	82
Avec les commandes SQL	85
Les commandes GRANT, REVOKE et DENY.....	88
La command GRANT, syntaxe simplifiée	88
Les roles creés par les utilisateurs. (pas ceux prédéfinis).	90
La commande REVOKE.	91
La commande DENY	91
Les vues pour la sécurité des données : contrôle sur les lignes.....	92
Conclusion	93
Le chiffrement des données.....	93

Définition :.....	93
Hachage « hashing » (chiffrement unidirectionnel).....	93
Chiffrement des données (chiffrement bidirectionnel)	94
Chiffrement des procédures et fonctions de la base de données	95
Chiffrer les données contenues dans une table	95
Chiffrement des données dans le SGBD MS SQL Server	95
Chiffrement des données dans le logiciel client ou le serveur d'application web	97
Autre exemple chiffrement par clé symétrique sans certificat.....	98
Autre exemple chiffrement par ENCRYPTBYPASSPHRASE	100
Sources	102

Historique des versions

Numéro de version	Tâches/modifications	Auteur	Date
1.0	Chapitres 1-6	Saliha Yacoub	Août 2019
1.1	Chapitre 7	Saliha Yacoub	Octobre 2019
1.2	Chapitre 8	Saliha Yacoub Marc Beaulne	Octobre 2019
1.3	Chapitre 9	Saliha Yacoub	Octobre 2019
1.4	Chapitre 10	Saliha Yacoub	Novembre 2019
1.5	Chapitre 10, le chiffrement	Marc Beaulne, Saliha Yacoub	Novembre 2019

Chapitre 1, pour bien commencer....

Microsoft SQL Server est un Système de gestion de base de données relationnel et transactionnel développé et commercialisé par Microsoft.

Microsoft SQL Server utilise le langage T-SQL (Transact-SQL) pour ses requêtes, c'est une implémentation de SQL qui prend en charge les procédures stockées et les déclencheurs. La dernière version est SQL Server 2017. La première ayant appartenu à Microsoft seul est en 1994. (Contrairement à Oracle qui sort la première version en 1979 voire 1977)

Durant, la session 2 nous avons étudié SQL en utilisant le SGBD Oracle. Il faut savoir que, tous les SGBDs relationnels (Oracle, MS SQL Server, MySQL, SQLite, DB2, PostgreSQL ..) utilisent un SQL standard.

Ce qui implique que TOUS ce que vous avez appris durant le cours de « Introduction aux bases de données » de la session 2 s'applique et reste valable pour les autres SGBDs à quelques exceptions près.

- La Commande CREATE TABLE reste la même. Mais certains SGBDs comme Oracle 12c et plus, MS SQL Server, et MY SQL ont implémenté le concept de l'incrément automatique de la clé primaire.
- La commande ALTER Table est la même. De même que la commande DROP Table.
- La commande SELECT reste la même. Les jointures se font au niveau du FROM et non au niveau du WHERE.
- Sauf le SQLite, les SGBD cités plus haut sont TOUS des SGBDS SERVEURS. SQLite est un SGBD embarqué.
- TOUS les SGBDs offrent une interface ou un logiciel de gestion des bases de données. Pour Oracle, nous l'avons vu, c'est SQL Developer. Pour MS SQL Server c'est **SQL Server Management Studio**, pour MySQL c'est MySQL Workbench, pour SQLite c'est SQLite DB Browser.

Cependant,

- Les SGBDs n'ont pas la même architecture. Pour Oracle, quelle que soit la version, il manque la couche « Base de Données ». Tous les usagers sont connectés à une unique base de données qui est ORCL dans la plupart des cas (sinon xe). La base de données est créée au moment de l'installation. Ce point est très important pour la suite du cours. Pour MS SQL Server, chaque utilisateur doit créer sa propre base de données, et il peut en créer plusieurs BD.

Lorsque vous êtes connectés à un serveur MS SQL Server, la première opération à exécuter est : (si vous n'avez pas de BD)

```
CREATE DATABASE nomde1aBD;
```

Exemple :

```
CREATE DATABASE emp1g;
```

Comme il est possible que vous ayez plus qu'une base de données, avant toute utilisation de celle-ci il faudra l'indiquer au SGBD.

```
USE nomde1aBD;
```

Exemple

```
USE emp1g;
```

Il est de même pour le SGBD MySQL concernant le CREATE DATABASE et le USE .

- Les attributs n'utilisent pas les mêmes types. Exemple, pour Oracle, on utilise le VARCHAR2(n) alors pour MS SQL server c'est le VARCHAR(n). Avant de créer une table, ce serait utile de consulter les types de données manipulés par le SGBD.
- MS SQL Server a implémenté la propriété **IDENTITY** pour l'incrémentation automatique de la clé primaire. Cette propriété se retrouve dans ORACLE 12c et plus. Pour MySQL, il utilise la propriété : AUTO_INCREMENT.
- Pour Oracle 11g (la base de données que nous avons utilisée lors de la dernière session), le principe de l'incrémentation automatique utilise une séquence et un trigger. On utilise nomSequence.nextval pour incrémenter automatiquement une valeur.

Attention :  (Rappel)

Si une séquence démarre à 1 pour Oracle 11g, la valeur qui sera insérée est 2. (nextval). Ce qui n'est pas le cas avec IDENTIT (1,1) qui indique que la valeur qui sera insérée est 1.

- Les SGBD sont très différents concernant l'extension de la couche SQL. Ils sont différents pour l'écritures des procédures stockées et des triggers. Cette session, nous allons étudier le **Transact-SQL** qui est l'extension du SQL pour MS SQL Server. À titre d'information, pour ORACLE, cette extension du SQL s'appelle : PL/SQL. Pour SQLite, cette couche gère uniquement les triggers.
- L'interface graphique de **SQL Server Management Studio** permet de créer directement la base de données à l'aide du schéma. En d'autres mots, pas besoin de générer le code SQL du diagramme pour l'exécuter puisque les tables sont déjà créées. Ce qui n'était pas le cas avec Oracle SQL developer Data Modeler où est-ce qu'il faut générer le code SQL puis l'exécuter. En ce sens, MY SQL WorkBench est semblable à Oracle.
- **SQL Server Management Studio** est un excellent outil pour créer et exploiter vos bases de données MS SQL Server indépendamment d'un langage de programmation. MAIS... il faut savoir que Visual Studio vous permet aussi de créer et gérer vos bases de données MS SQL Server. On verra ce point plus loin.
- Ce qu'il faut savoir pour la suite du cours, c'est que votre poste de travail est à la fois serveur et client. Pas comme l'installation qu'on avait avec Oracle. Dans ce cas, il faut être conscient que n'importe qui peut supprimer votre BD puisque tous les étudiants sont ADMIN de leur poste de travail. Par conséquent il faut :
 - Essayer le plus possible de garder votre poste de travail le reste de la session.
 - Garder en tout temps vos scripts SQL.

Chapitre 2, installation, configuration et connexion

Si vous n'avez pas déjà installé SQL server, vous devez le faire. L'installation de la base de données est très simple et se fait automatiquement.

Nous avons besoin d'installer :

- 1- Le serveur de bases de données :

Vous devez aller sur le site suivant pour télécharger et installer SQL Server Express 2017.

<https://www.microsoft.com/fr-ca/sql-server/sql-server-editions-express>

Vous devez choisir installation Standard, et tout se déroule automatiquement.


Attention ! vous devez vérifier les paramètres de langue de votre ordinateur.

- 2- L'outil de gestion de bases de données

Une fois que le serveur est installé, vous devez installer SSMS version 18.2 (SQL Server Management Studio), ce qui vous permet de gérer et d'exploiter vos bases de données avec SQL Server. Pour cela vous devez vous rendre sur le site :

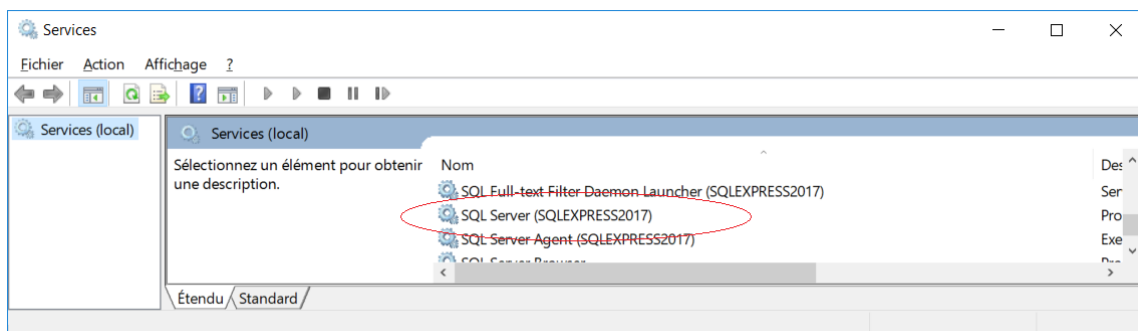
<https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017>

L'installation se fait automatiquement.

Attention : 

Il faut redémarrer l'ordinateur pour que l'installation soit complète

Si votre serveur ne démarre pas, il faudra le faire manuellement :



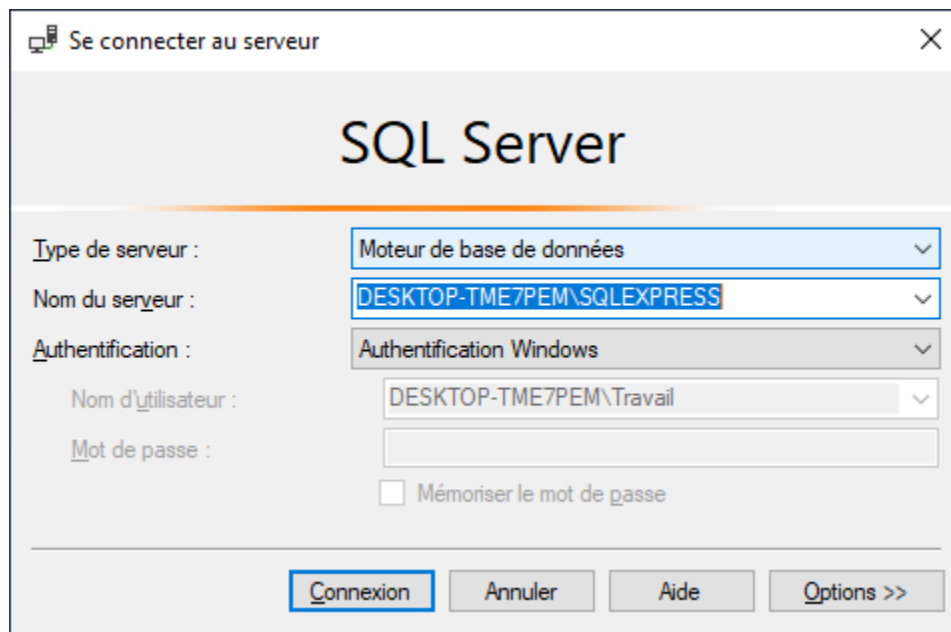
Mode d'authentification :

Authentification Windows : Si vous choisissez ce mode d'authentification, cela veut dire que le serveur de base de données, lorsque vous essayez de vous connecter, ne vous demandera pas de mot de passe. Utiliser ce mode d'authentification si vous n'avez pas de compte sur le serveur de base de données. C'est avec ce mode que l'on se connecte pour la première fois.

Authentification SQL server : Si vous choisissez ce mode d'authentification, cela veut dire que vous avez un compte sur le serveur de bases de données. Vous avez besoin d'un nom d'utilisateur, d'un mot de passe et d'une base de données. C'est ce mode d'authentification que l'on va utiliser durant toute la session. C'est ce mode d'authentification que vous allez avoir en entreprise.

Étape 1 : Changer le mode d'authentification


Lorsqu'on établit une connexion pour la première fois, nous allons faire une authentification Windows. (Vous n'avez pas encore de compte sur le Serveur SQL server) — voir la figure suivante.



The screenshot shows a Windows dialog box titled "Se connecter au serveur" (Connect to server) for "SQL Server". The dialog contains the following fields and options:

- Type de serveur : Moteur de base de données (dropdown menu)
- Nom du serveur : DESKTOP-TME7PEM\SQLEXPRESS (dropdown menu)
- Authentification : Authentification Windows (dropdown menu)
- Nom d'utilisateur : DESKTOP-TME7PEM\Travail (dropdown menu)
- Mot de passe : (empty text field)
- Mémoriser le mot de passe (checkbox)

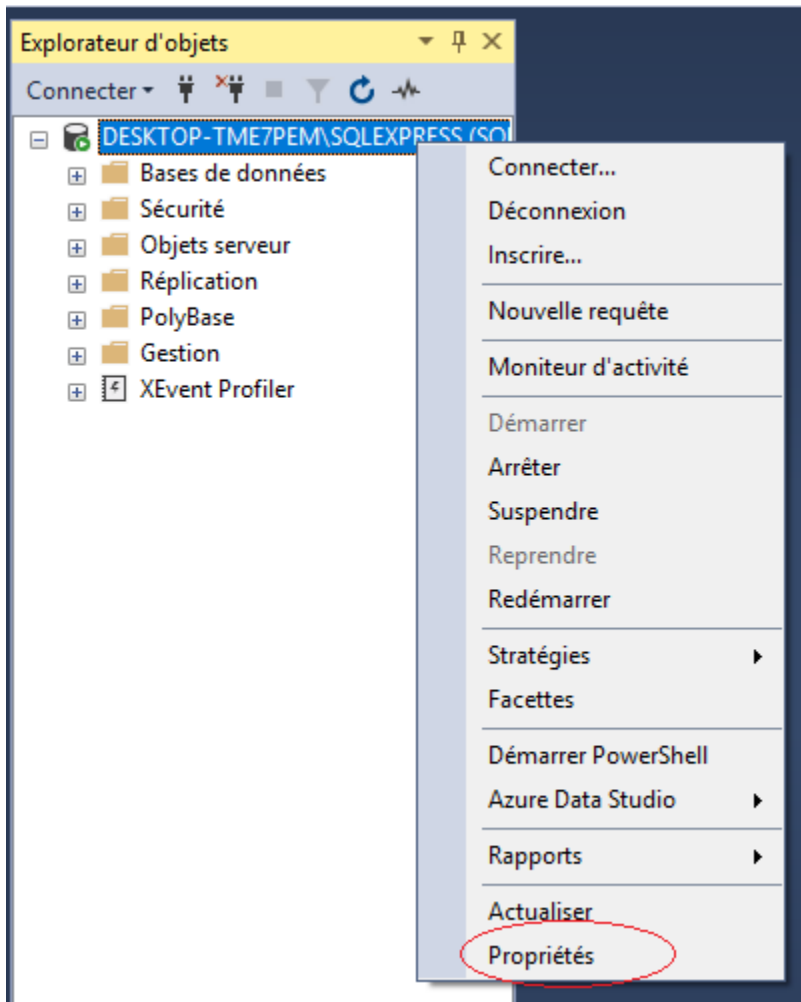
At the bottom, there are four buttons: "Connexion", "Annuler", "Aide", and "Options >>".

Attention : 

Le nom du serveur est le nom de votre ordinateur\nom de l'instance

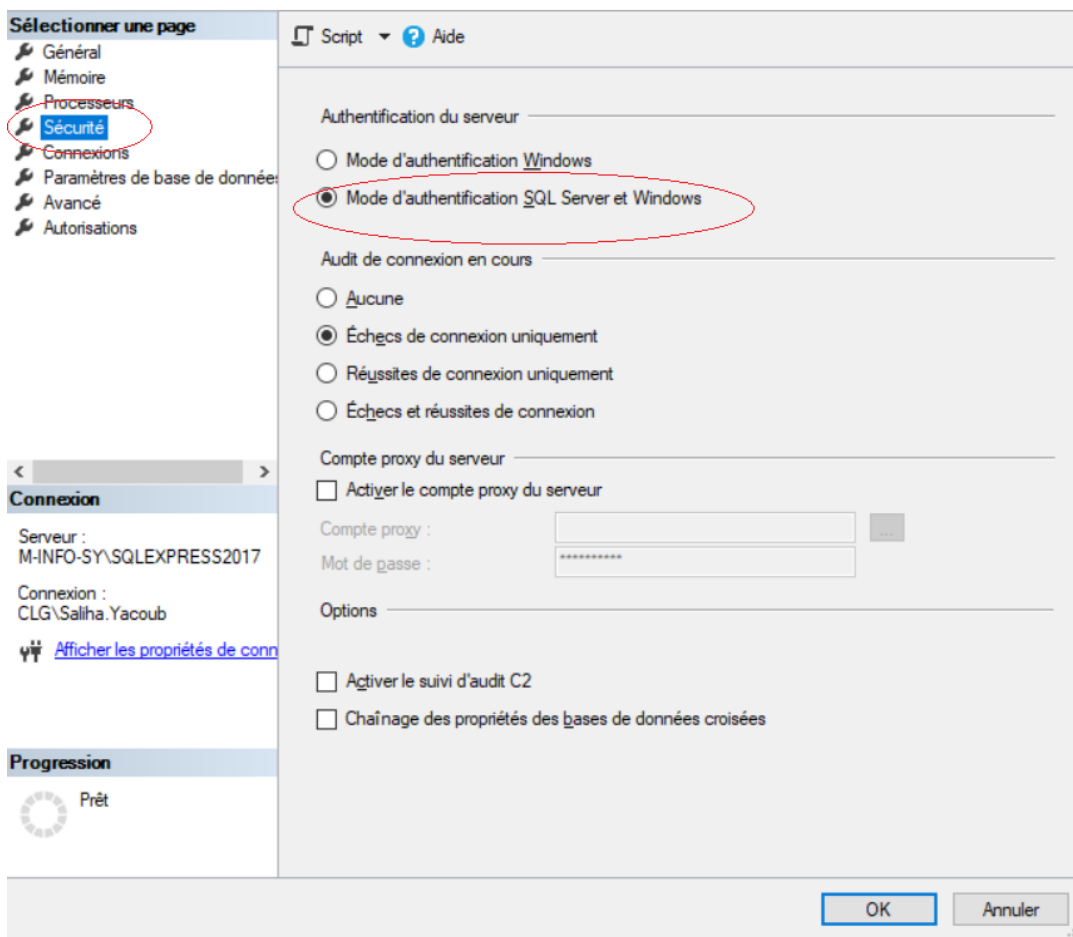
Sinon, dérouler le nom du serveur, faire parcourir (ou <Browse for more ...> et trouver votre serveur et son instance.

Une fois que vous êtes connecté, allez sur les propriétés de votre connexion et changez le mode d'authentification. → Figure suivante.



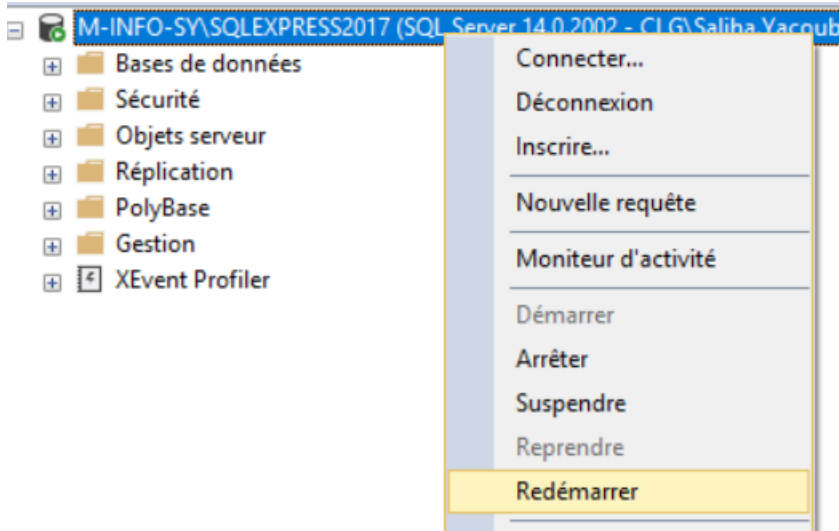
Bouton droit sur votre serveur, puis propriété Sécurité

A l'onglet Sécurité, choisir Mode d'authentification SQL Server et Windows. Faites OK. Redémarrer le serveur. (Bouton droit puis redémarrer.).

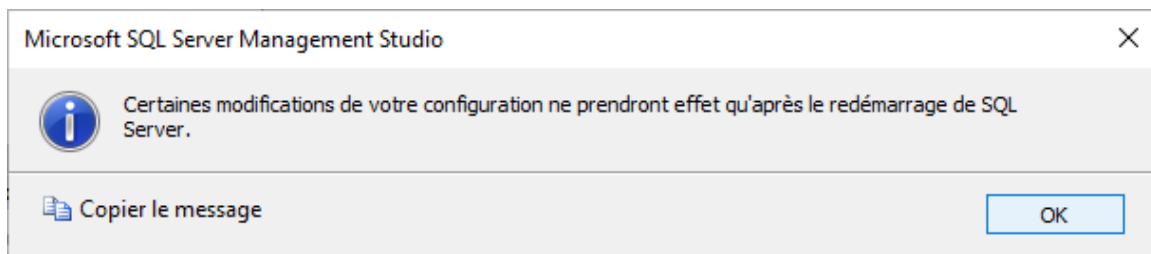


Attention : 

Vous devez redémarrer le serveur

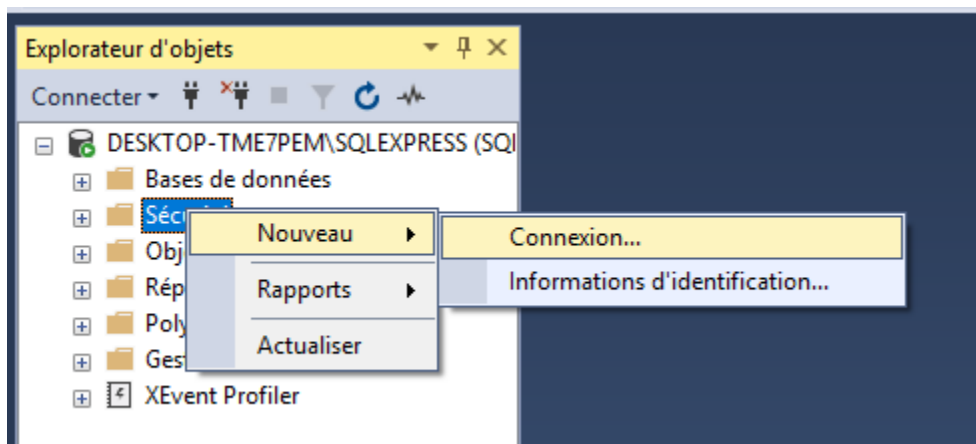


Il est probable que le serveur vous fasse une mise en garde quant au changement du mode d'authentification et qu'il faut redémarrer le serveur. Faîtes juste OK.



Étape 2 : Créer une nouvelle connexion

Sur le bouton droit de l'onglet **Sécurité**, créer une nouvelle connexion.



- Donner un nom significatif sans caractères spéciaux et sans accents
- Choisir Authentification SQL Server.
- Choisir un mot de passe qui respecte la stratégie des mots de passe Windows Server

Sélectionner une page

- Général
- Rôles du serveur
- Mappage d'utilisateur
- Éléments sécurisables
- État

Script ? Aide

Nom d'accès : Rechercher...

Authentification Windows
 Authentification SQL Server

Mot de passe :

Confirmer le mot de passe :

Spécifier l'ancien mot de passe

Ancien mot de passe :

Appliquer la stratégie de mot de passe
 Appliquer l'expiration du mot de passe
 L'utilisateur doit changer de mot de passe à la prochaine connexion

Mappé au certificat
 Mappé à la clé asymétrique
 Mapper aux informations d'identification

Informations d'identification mappées

Informations ...	Fournisseur

Ajouter

Supprimer

Base de données par défaut :

Langue par défaut :

Connexion

Serveur : DESKTOP-TME7PEM\SQLEXP

Connexion : DESKTOP-TME7PEM\Travail

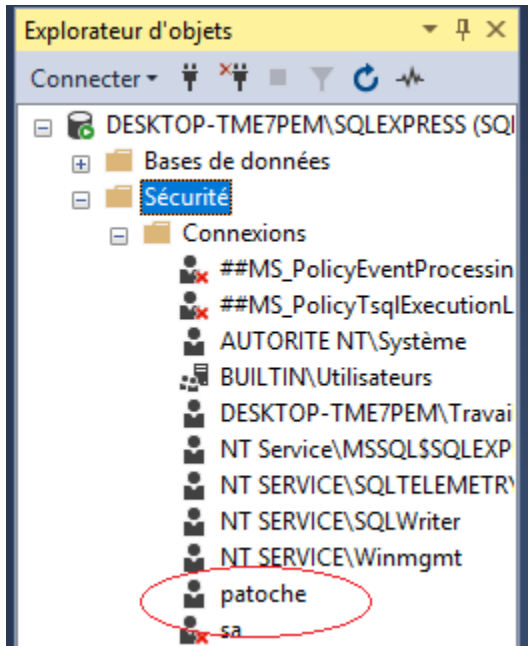
[Afficher les propriétés de connexion](#)

Progression

Prêt

- Décocher l'utilisateur doit changer le mot de passe.
- Vous pouvez décocher la case « Conserver la stratégie des mots de passe. Mais ce n'est pas conseillé.
- Comme vous n'avez pas de base de données, la connexion utilise la Base de données par défaut qui **master**.
- Ne vous inquiétez pas, vous aller avoir votre propre base de données


Une fois que cette étape est terminée, vérifiez que votre connexion est bien créée. Pour cela allez dans l'onglet Sécurité-puis Connexion et repérez votre connexion



Étape 3 : Attribuer les rôles

Pour pouvoir créer votre propre base de données vous devez posséder les droits nécessaires (ou le rôle).

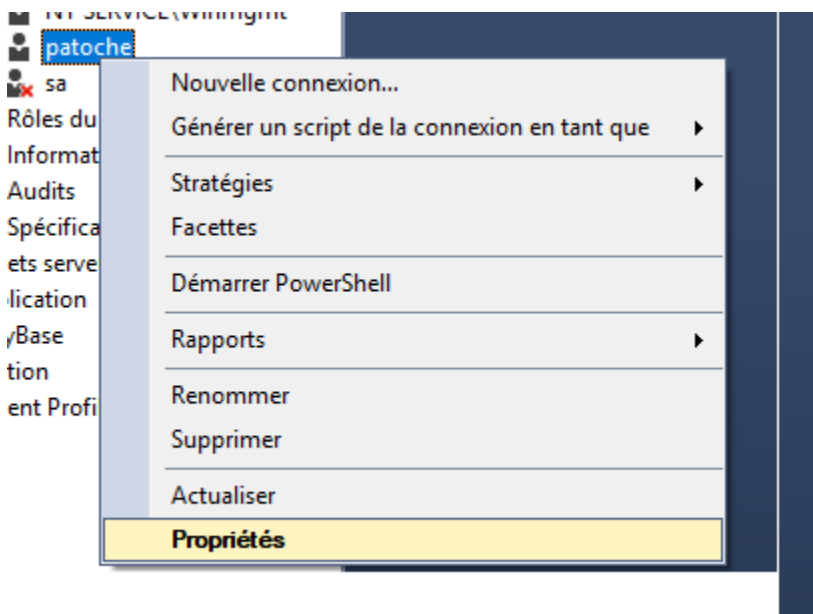
Si vous êtes administrateur alors vous avez déjà ces rôles, sinon vous devez les attribuer à votre connexion avant de créer la bd.

Attention : 

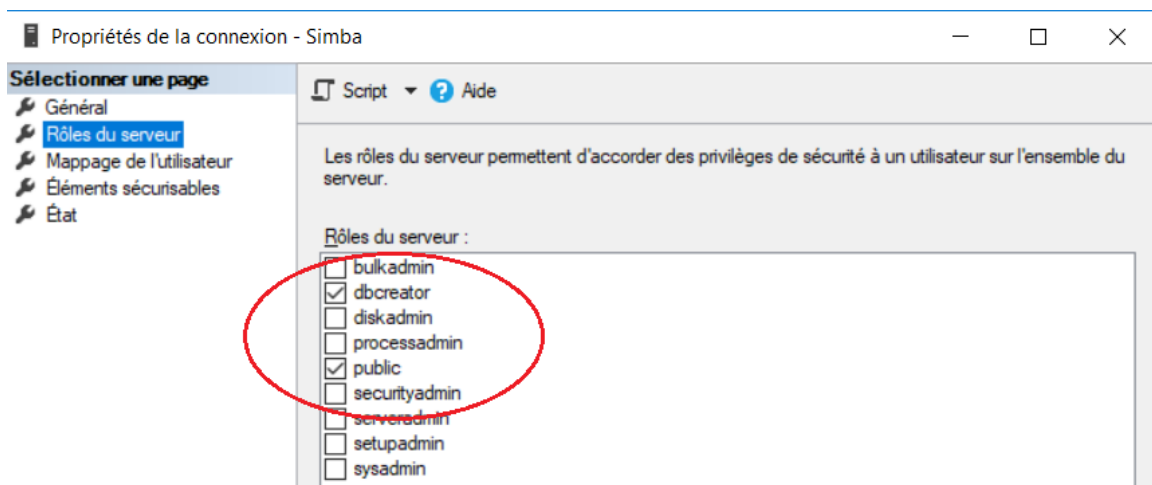
Pour créer la base de données vous devez avoir au moins le rôle **dbcreator**

Les membres du rôle de serveur **dbcreator** peuvent créer, modifier, supprimer et restaurer n'importe quelle base de données.

Pour donner les droits à votre connexion, allez à votre connexion, bouton droit, propriétés puis rôle du serveur



Puis Rôles du serveur.



Puis cocher **dbcreator** puis cliquer sur OK.

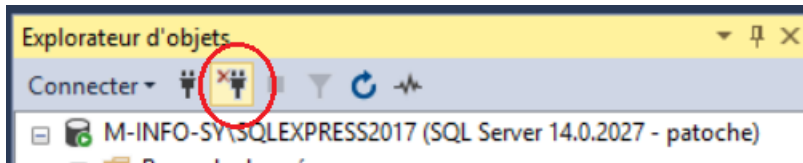
Attention :

Ne jamais donner le rôle sysadmin. Les membres du rôle sysadmin peuvent effectuer toute activité sur le serveur. Faites attention !!


Étape 4, Connexion avec l'authentification SQL Server et création de la base de données.

Vous pouvez vous déconnecter du serveur et vous reconnecter avec votre nouvelle connexion (SQL Server) comme suit.

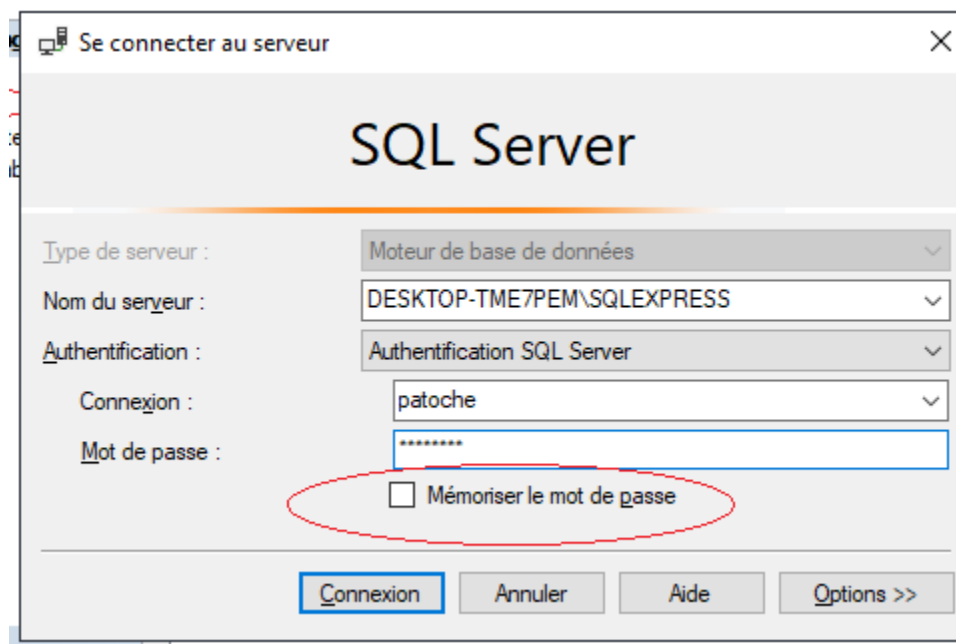
Pour vous déconnecter du serveur, utiliser le bouton Déconnecter




Ou bien Bouton droit sur la Votre serveur, puis **déconnecter** .

Attention : 

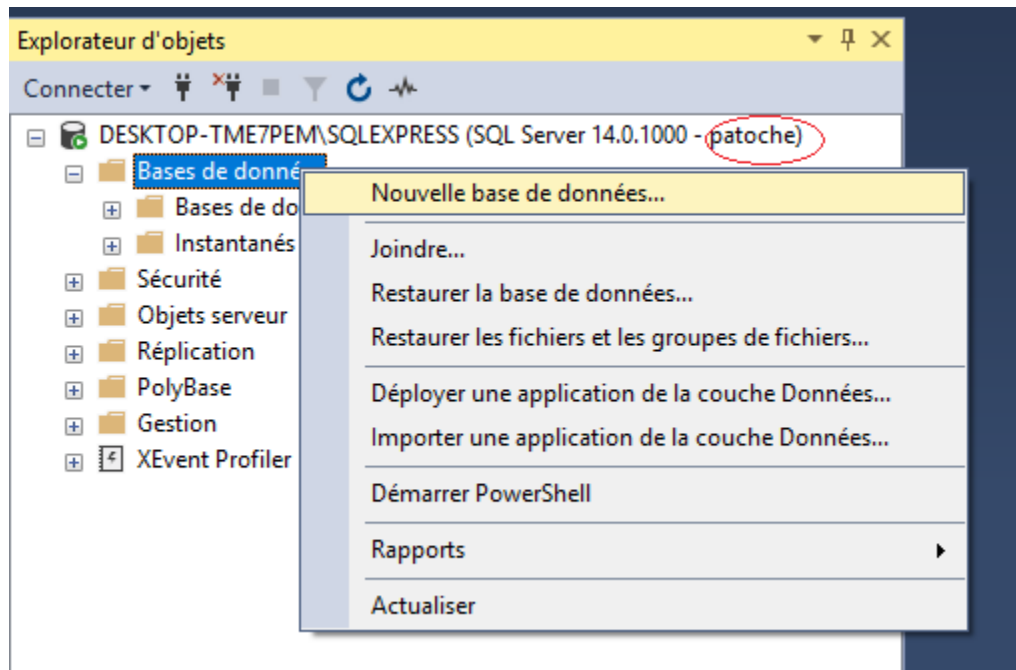
Ne jamais mémoriser le mot de passe



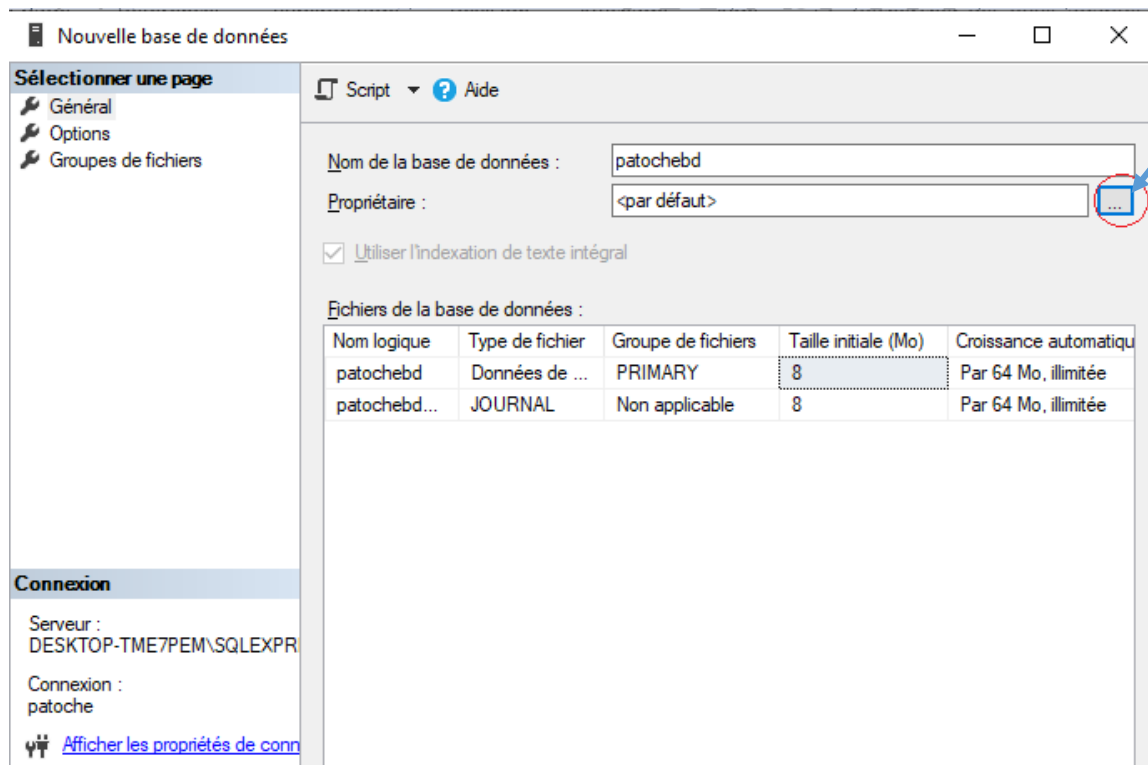
Attention : 

Ne jamais mémoriser le mot de passe

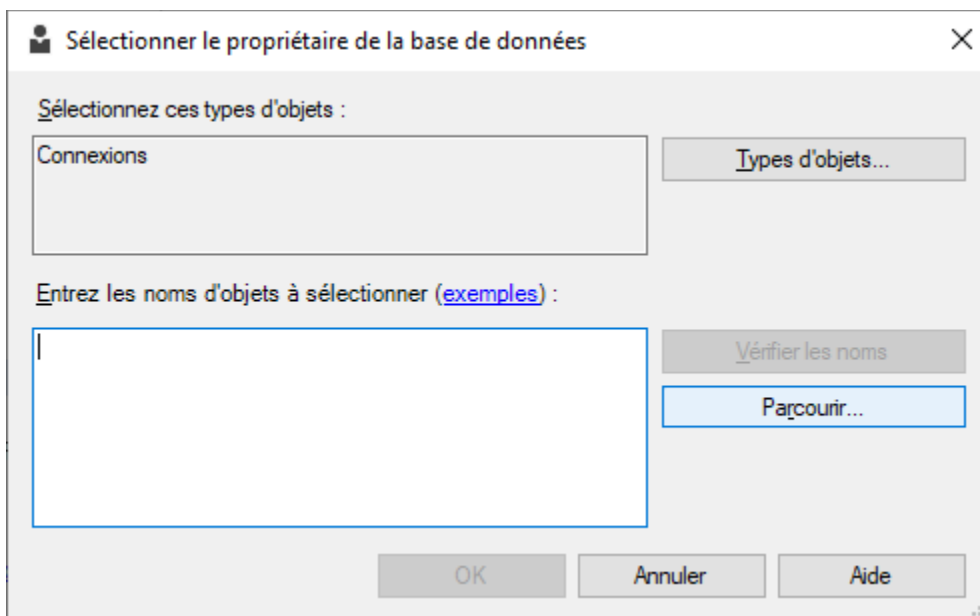
Pour créer une nouvelle base de données, placez-vous à l'onglet bases de données, puis nouvelle base de données. Ou utiliser la commande CREATE DATABASE



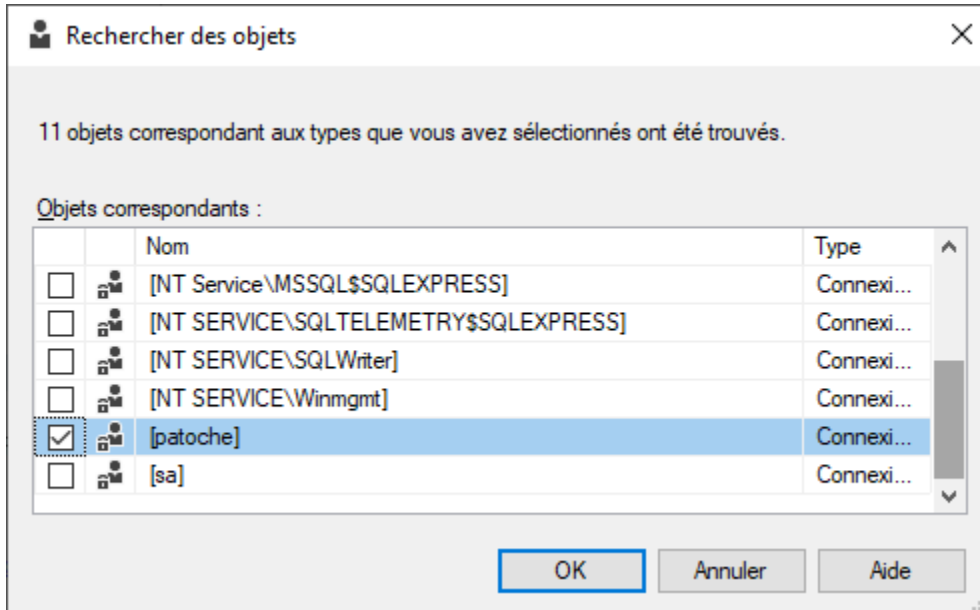
Donnez un nom significatif à votre base de données.



Avant de cliquer sur OK, cliquer sur propriétaire, vous allez avoir la figure suivante :



Cliquez ensuite sur parcourir, puis trouvez votre connexion et cochez-la. (voir figure suivante).



Cliquez OK sur chaque fenêtre

Après la création de la base de données, nous allons faire en sorte que le login pointe directement sur la nouvelle base de données.

Sous l'onglet Sécurité, déroulez les connexions. Repérez la vôtre. Puis bouton droit de la souris et Propriétés.

Roles du serveur
Mappage d'utilisateur
Éléments sécurisables
État

Connexion

Serveur :
DESKTOP-TME7PEM\SQLEXPRESS

Connexion :
patoche

Afficher les propriétés de connexion

Progression

Prêt

Nom d'accès : patoche Rechercher...

Authentification Windows

Authentification SQL Server

Mot de passe :

Confirmer le mot de passe :

Spécifier l'ancien mot de passe

Ancien mot de passe :

Appliquer la stratégie de mot de passe

Appliquer l'expiration du mot de passe

L'utilisateur doit changer de mot de passe à la prochaine connexion

Mappé au certificat

Mappé à la clé asymétrique

Mapper aux informations d'identification

Informations d'identification mappées

Informations ...	Fournisseur

Base de données par défaut : patochebd

Langue par défaut :

Supprimer

Annuler

Choisir ensuite le nom de votre BD par défaut. Tester à nouveau votre connexion.

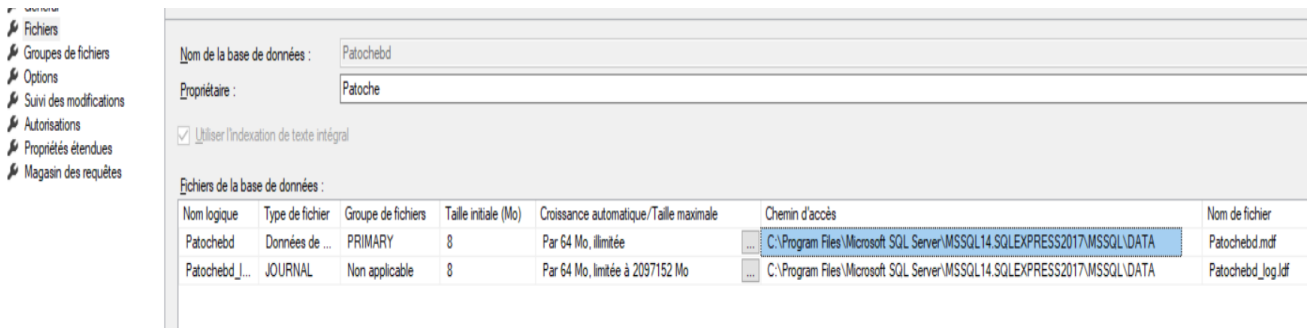
Important :

Vous pouvez également créer votre base de données avec la commande CREATE

```
CREATE DATABASE nomdeLaBD;
```

Où est stockée la base de données ?

En cliquant sur le bouton droit de votre base de données, puis propriétés à l'onglet fichier vous allez trouver les deux fichiers de la bd et leur emplacement.



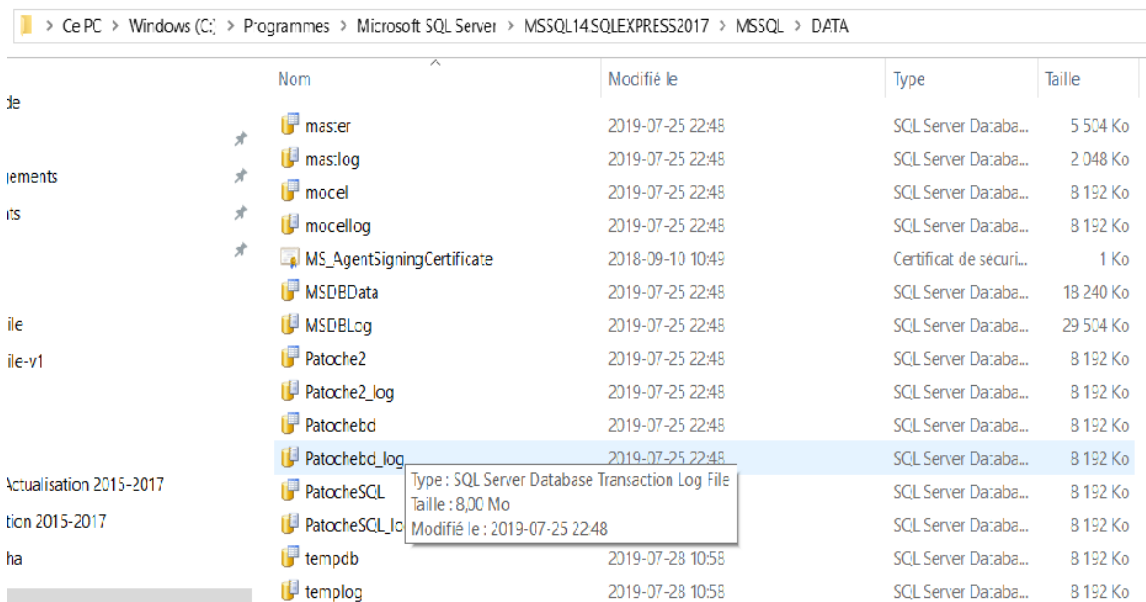
Ces fichiers sont dans :

C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS2017\MSSQL\DATA

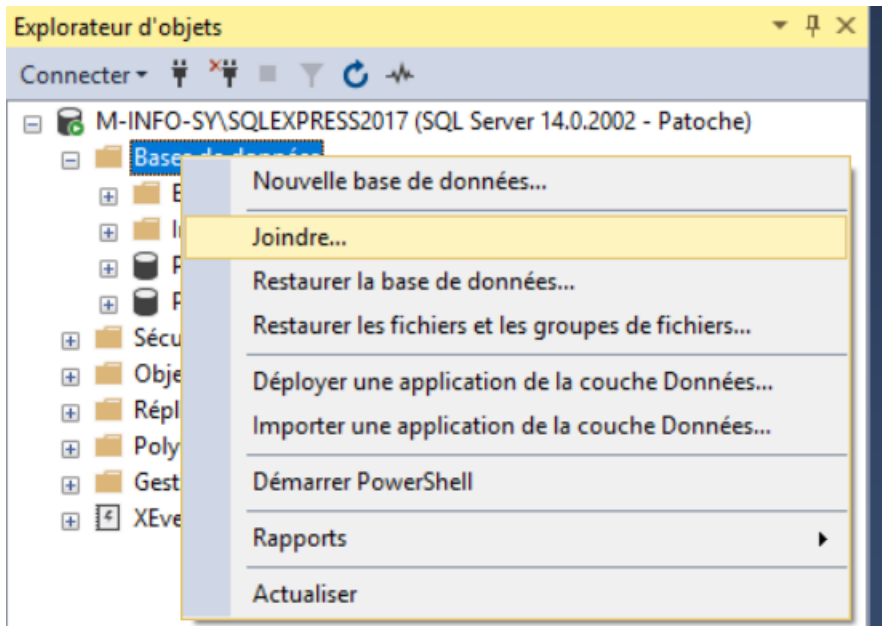
Vous y trouverez deux types de fichiers pour chaque BD : L'un **.mdf** et l'autre **.ldf**

Les données sont stockées dans un fichier MDF, toutes les transactions, les modifications de la base de données SQL Server effectuées par chaque transaction sont stockées dans un fichier LDF

Patochebd.mdf et Patochebd_log.ldf

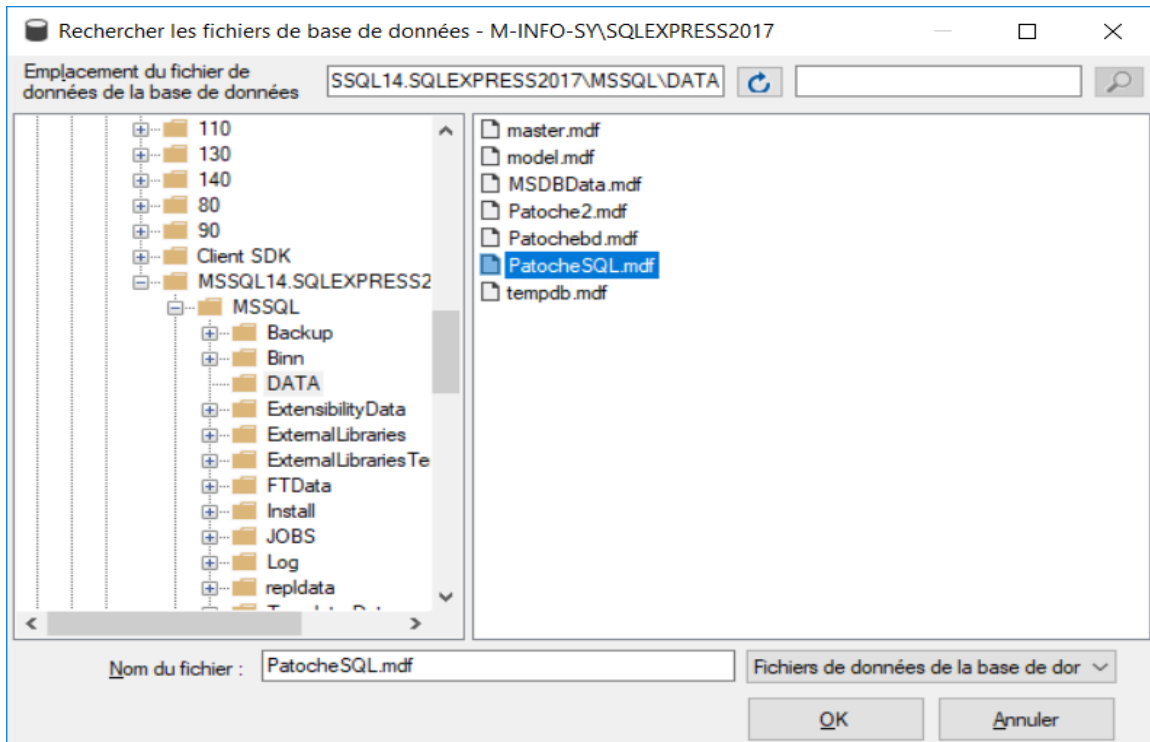


Vous pouvez récupérer votre base en faisant : Bouton droit sur Bases de données, puis **Joindre**. Vous aurez la fenêtre suivante.



Cliquer sur le bouton Ajouter. Choisir le fichier en question (le fichier.mdf) puis faire OK, puis OK.

Votre base de données va apparaître dans l'explorateur d'objets. Vous pouvez alors l'exploiter comme vous voulez.




Attention : 

Une base de données ne peut être jointe plus qu'une fois.

Lorsque vous essayez de joindre une Base de données déjà jointe, cela provoquera une erreur.

Il faut que vos fichiers soient dans le bon dossier.

Attention Récupération de la base de données: 

Pour récupérer votre base de données effectuer les étapes suivantes :

1- Dans tous les cas garder vos scripts SQL.

2-Copier les deux fichiers .mdf et .ldf de votre base de données (patochebd et patoche_log) dans votre clé USB

3-pour ouvrir les fichiers que vous avez copiés dans votre clé USB sur un autre ordinateur, vous devez d'abord JOINDRE le fichier.mdf

4-Si vous êtes certains que votre BD a été copiée proprement alors vous pouvez la supprimer pour qu'il n'y ait pas de plagiat.

Chapitre 3, création des tables

Types de données SQL Server

Types numériques exacts

Type	À partir de	À
bigint	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
int	-2147483648	2147483647
smallint	-32768	32767
tinyint	0	255
bit	0	1
Decimal	-10^{38}	$10^{38}-1$
numeric	-10^{38}	$10^{38}-1$
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Numerics approximatif

Type	À partir de	À
float	$1,79 \text{ E} + 308$	$1,79 \text{ E} + 308$
reel	$-3.40\text{E} + 38$	$3.40\text{E} + 38$

datetime et smalldatetime

Type	À partir de	À
datetime (3,33 exactitude millisecondes)	1 janvier 1753	31 déc 9999
smalldatetime (précision de 1 minute)	1 janvier 1900	6 juin 2079

Chaînes de caractères

Type	Description
char	De longueur fixe de caractères Unicode avec une longueur maximum de 8000 caractères.
varchar	Texte unicode de longueur variable allant jusqu'à 2 Go.
text	Texte non unicode de longueur maximale 2Go

Les chaînes de caractères Unicode

Type	Description
nchar	la longueur de données Unicode-fixe avec une longueur maximale de 4000 caractères.
nvarchar	la longueur de données Unicode et variable, avec une longueur maximum de 4000 caractères.
nvarchar (max)	longueur Unicode données variables avec une longueur maximale de 230 caractères (SQL Server 2005 uniquement).
ntext	la longueur de données Unicode et variable, avec une longueur maximale de 1073741823 caractères.

Binary Cordes

Type	Description
binaire	De longueur fixe des données binaires d'une longueur maximale de 8000 octets.
varbinary	De longueur variable des données binaires d'une longueur maximale de 8000 octets.
varbinary (max)	De longueur variable des données binaires d'une longueur maximale de 231 octets (SQL Server 2005 uniquement).
image	De longueur variable des données binaires d'une longueur maximale de 2147483647 octets.

Pour plus de détails, allez sur :

<https://docs.microsoft.com/fr-fr/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-2017>

La propriété « IDENTITY » d'une table

Vous pouvez mettre en œuvre des colonnes d'identification à l'aide de la propriété IDENTITY. Ce qui permet de réaliser un auto incrément sur une colonne.

En général, la propriété IDENTITY se définit sur la clé primaire.

Si la propriété IDENTITY est définie sur une colonne, alors c'est le système qui insère des données dans cette colonne (pas l'utilisateur).

Attention : 

Vous ne pouvez pas modifier une colonne existante pour y ajouter la propriété IDENTITY.

Exemple1 :

```
create table Eleves
(
  num int identity,
  nom varchar(30),
  prenom varchar(30),
  constraint pkeleve primary key(num));

insert into Eleves (nom,prenom) values('Patoche','Alain');
insert into Eleves (nom,prenom) values('Simba','Chat');
```

Remarquez :

1. La colonne num a la propriété IDENTITY.
2. Nous n'avons pas inséré dans la colonne num. C'est le système qui le fait pour nous.
3. Le num de Patoche sera 1, le num de Simba sera 2 et ainsi de suite.

Lorsque vous utilisez la propriété IDENTITY pour définir une colonne d'identification, tenez compte des éléments suivants :

- Une table ne peut comprendre qu'une colonne définie à l'aide de la propriété IDENTITY, et cette colonne doit être définie à l'aide d'un type de données **decimal**, **int**, **numeric**, **smallint**, **bigint** ou **tinyint**.
- Vous pouvez spécifier la valeur de départ et l'incrément. La valeur par défaut est 1 dans les deux cas.
- La colonne d'identification ne doit ni accepter les valeurs NULL, ni contenir une définition ou un objet DEFAULT. **En général c'est la clé primaire.**
- La colonne peut être référencée dans une liste de sélection par l'emploi du mot clé \$IDENTITY après la définition de la propriété IDENTITY. La colonne peut également être référencée par son nom.
- SET IDENTITY_INSERT ON peut être utilisé pour désactiver la propriété IDENTITY d'une colonne en activant les valeurs à insérer explicitement.

Exemple 2

La numérotation automatique commence à 10 et elle est à pas de 2.

```
create table ClientsInfo
(
  numcl smallint IDENTITY(10,2),
  nomcl varchar(30),
  constraint pkcl primary key (numcl)
);

insert into Clientsinfo (nomcl) values ('Gavroche');
```

Si on veut faire une insertion manuelle dans la colonne num (IDENTITY) il faut mettre IDENTITY_INSERT à ON

Exemple 3 :

```
set identity_insert eleves on;

insert into eleves (num,nom, prenom) values
(20, 'Simpson', 'Fred');
```

Attention, le num de Simpson est 20

Si on veut revenir à l'incréméntation automatique il faut faire

```
set identity_insert eleves off;

insert into eleves(nom, prenom) values ('Simon', 'Pascal');
```

Attention: le num de Simon est 21.

Création des tables avec SQL Server

Les mêmes syntaxes s'appliquent à la création de tables avec SQL Server.

Exemple1 : Dans l'exemple qui suit remarquez la colonne **identity** et le type **money**

```
create table programmes
(codep char(3),
nomprogramme varchar(30),
constraint pkprg primary key(codep)
);

create table etudiants
(
numad int identity ,
nom varchar(20),
prenom varchar(30),
salaire money,
codep char(3),
constraint fkprg foreign key(codep)
references programmes(codep),
constraint pktudiant primary key(numad)
);
```

Remarque1 : Avec SQL Server on peut définir un seul INSERT INTO et donner la liste des valeurs.

```
insert into programmes values
('inf', 'Informatique'),
('tge', 'Technique de genie'),
('ele', 'Electronique'),
('sim', 'Sciences maths info');
```

Pour exécuter un commit après une opération DML, il faut mettre l'opération entre

```
begin transaction;
opérations DML
commit;
```

Exemple 2

```
begin transaction;  
insert into etudiants (nom, prenom, salaire, codep) values  
( 'Patoche', 'Alain', 12.33, 'tge' );  
insert into etudiants (nom, prenom, salaire, codep) values  
( 'Gavroche', 'Miserable', 1.33, 'inf' );  
insert into etudiants (nom, prenom, salaire, codep) values  
( 'Bien', 'Henry', 18.33, 'inf' );  
insert into etudiants (nom, prenom, salaire, codep) values  
( 'Leriche', 'Alain', 40.00, 'inf' );  
commit;
```

En général, l'ensemble des contraintes que nous avons vues avec ORACLE se définissent de la même façon avec SQL Server.

```
alter table etudiants add constraint  
cksal check(salaire>1);
```

Chapitre 4, le modèle de données avec SQL Server Management Studio.

Parfois, il est intéressant, même très utile de concevoir le modèle de la base de données (modèle relationnel) puis de générer le code SQL. C'est le cas de la plupart des SGBD. On se souvient par exemple du SQL Data Modeler du SGBD Oracle.

Pour MS SQL Server, c'est très simple de créer la base de données en utilisant un schéma relationnel.

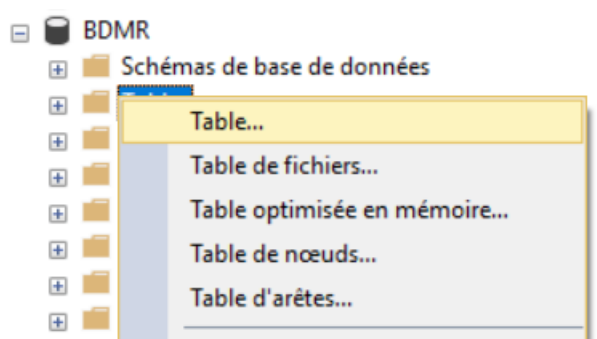
Vous pouvez soit obtenir un modèle relationnel d'une base de données déjà créée ou tout simplement créer un nouveau schéma.

Étape 0 : création de la base de données

Créer une nouvelle base de données avec un nom significatif (ou votre nom) Faites en sorte que vous en soyez le propriétaire.

Étape 2 : Création des tables :

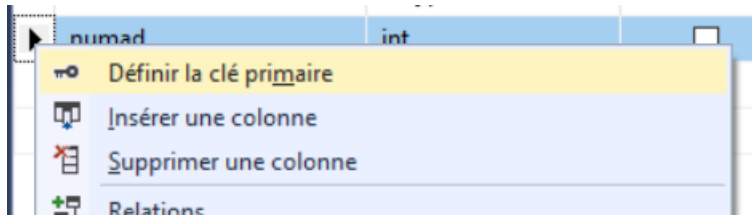
- 1- Faire bouton droit de sur l'onglet Tables de votre BD, puis table



- 2- Créer une table avec les colonnes souhaitées. Les types de données sont ceux que nous avons au chapitre 3

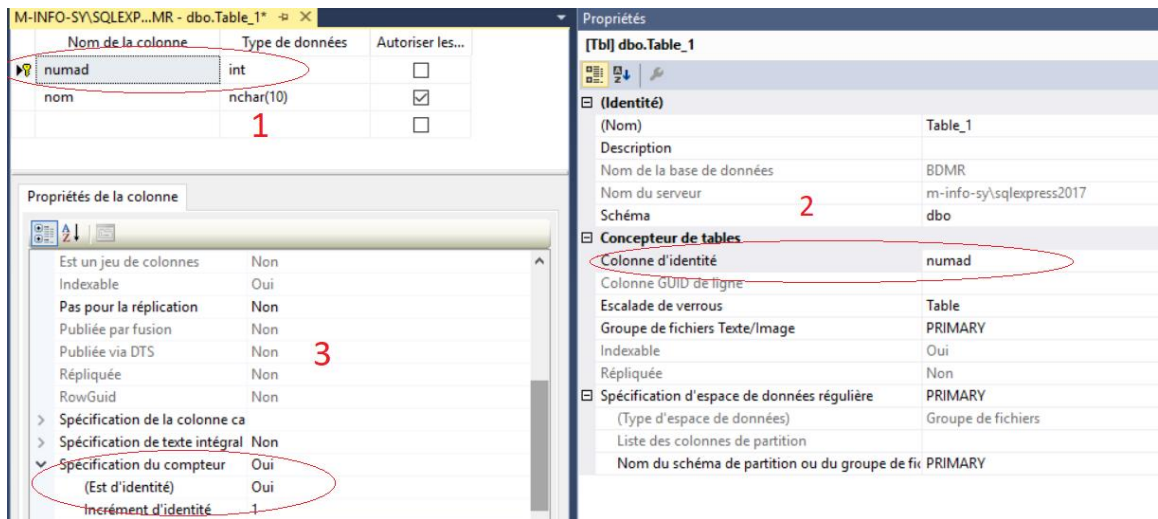
	Nom de la colonne	Type de données	Autoriser les...
?	numad	int	<input type="checkbox"/>
	nom	varchar(50)	<input checked="" type="checkbox"/>
	codep	char(3)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

- 3- Sélectionner la colonne de vous voulez qu'elle soit clé primaire, puis bouton droit et faire : définir la clé primaire : cette étape est obligatoire si vous voulez que la BD soit en 1FN.



Si vous voulez que votre clé primaire soit définie comme IDENTITY, alors :

- a. Positionnez-vous à la colonne de la clé primaire → 1
- b. Vérifiez que dans les propriétés de cette colonne, (à gauche → 2) la propriété « colonne d'identité » soit la clé primaire.
- c. Par la suite, vous allez remarquer que la propriété IDENTITY est bien définie sur la colonne → 3

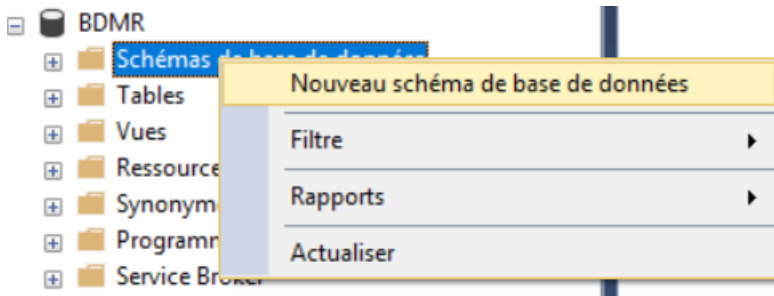


- 4- Enregistrer la table : cliquez sur le bouton enregistrez, puis donnez un nom à votre table. Notre table a pour nom : **EtudiantsInfo**

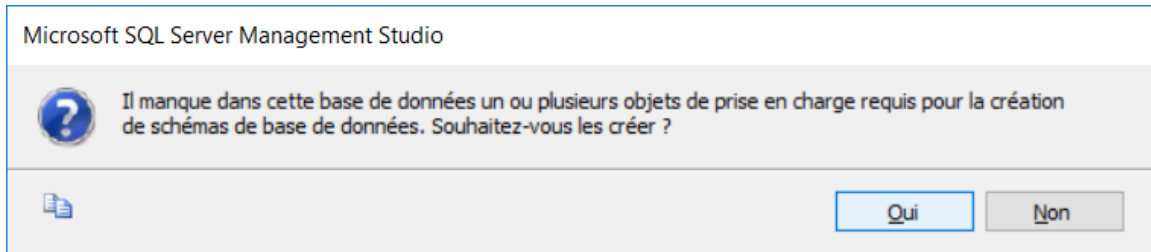
Étape 3, créer le schéma de la BD

Cette étape peut se faire après avoir créé l'ensemble des tables, ou après avoir créé la première table.

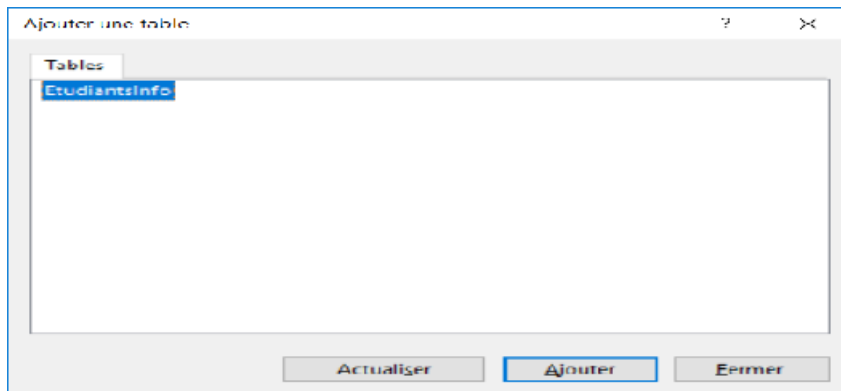
1. Sur le bouton droit de la BD, faire nouveau schéma :



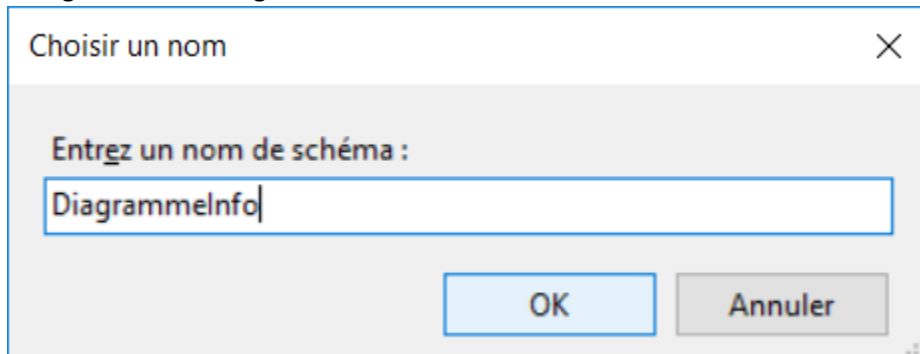
2. Si vous avez ce message, faites OK



3. Ajouter ensuite les tables à votre schéma. Pour l'instant la seule table que nous avons est EtudiantsInfo



4. Enregistrez votre diagramme.



Étape 4 : Définir les relations (la clé étrangère)

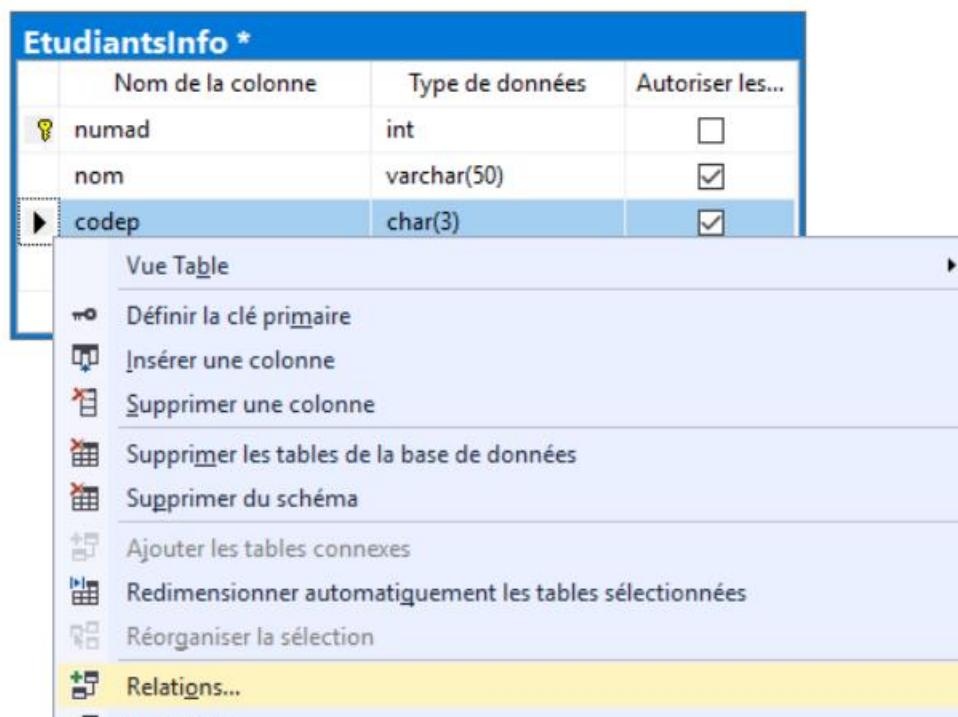
Une fois que vos tables sont créées, ou bien au fur et à mesure que vos tables vont se créer, il sera important de définir les liens entre les tables. Ces liens sont évidemment définis par le concept de Foreign Key ou clé étrangère.

Il est important de rappeler que les types de données et la taille des colonnes qui définissent la clé primaire et la clé étrangère soient les mêmes.

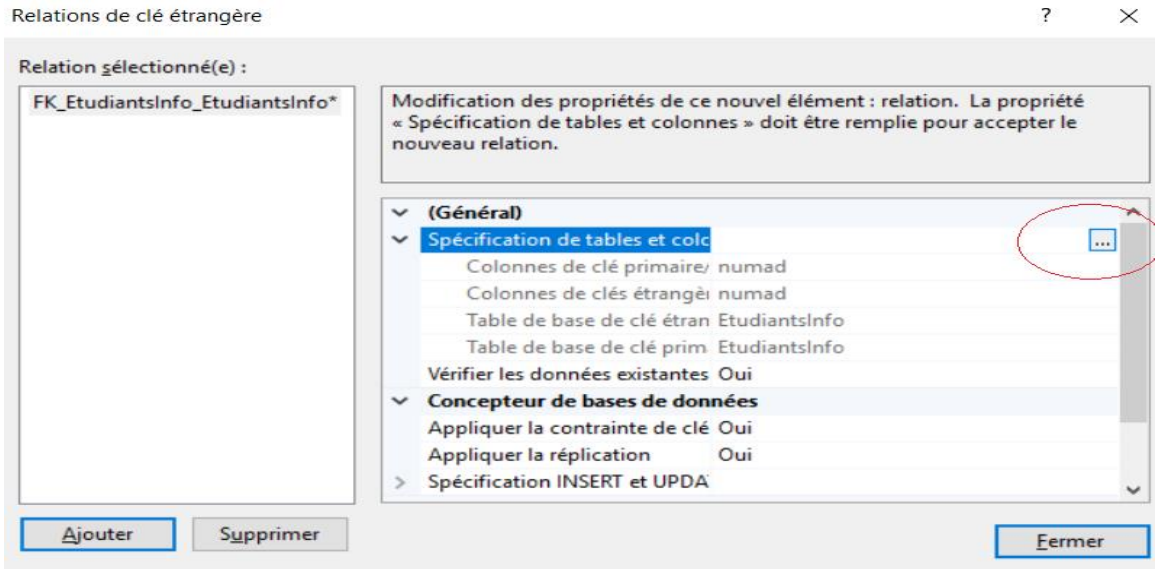
On suppose que la table ProgrammesInfo est créée.

PogrammesInfo *			
	Nom de la colonne	Type de données	Autoriser les..
🔑	codep	char(3)	<input type="checkbox"/>
	nomprog	varchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

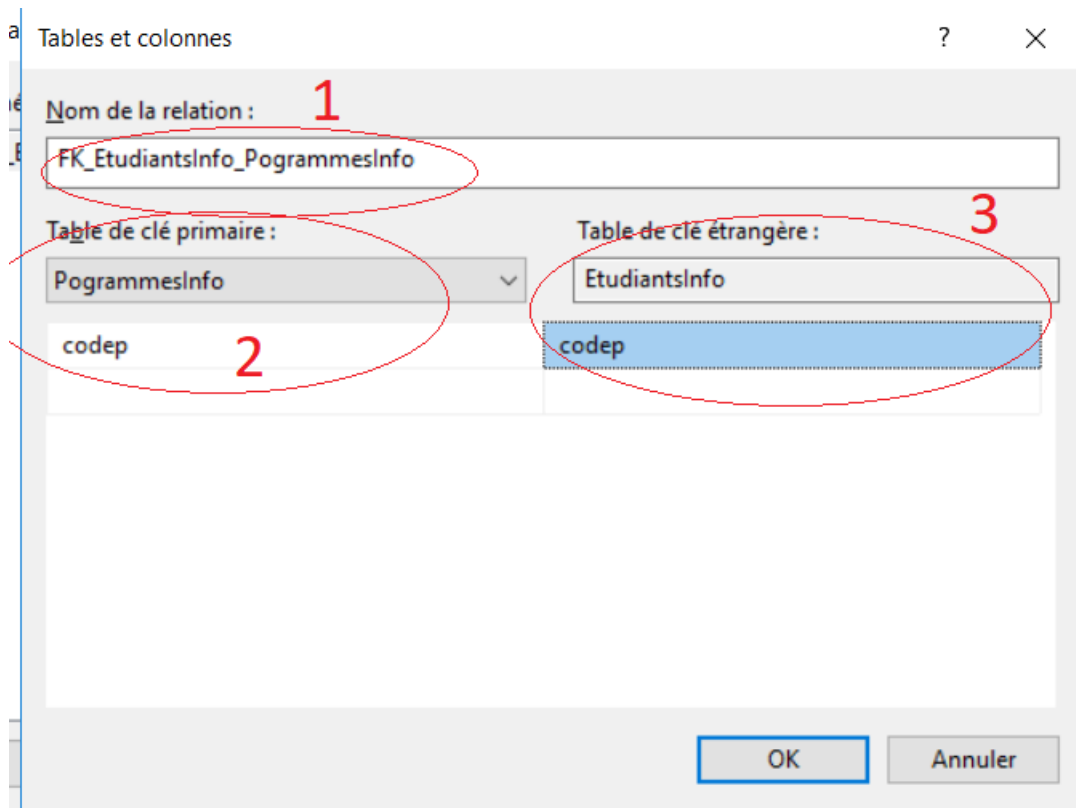
1. Sur la colonne Codep de EtudiantsInfo, (la colonne qui sera clé étrangère), faire Relation comme le montre la figure




2. Une fenêtre s'ouvre, faire Ajouter. Une fenêtre s'ouvre.
3. Dérouler, spécification des tables et des colonnes



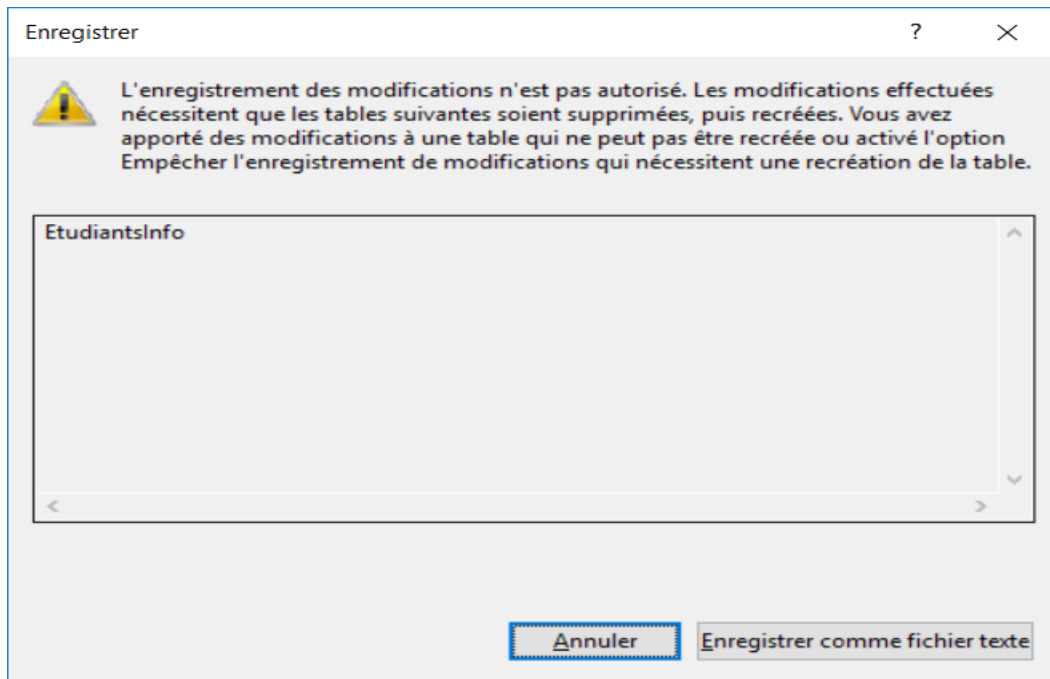
4. Vérifiez que vous avez bel et bien les bonnes colonnes avec les bonnes tables :
 - a. Vous pouvez changer le nom de la contrainte de FK →1
 - b. Vérifier la table et la colonne de la primary Key →2
 - c. Vérifier la table et la colonne de la FK→3



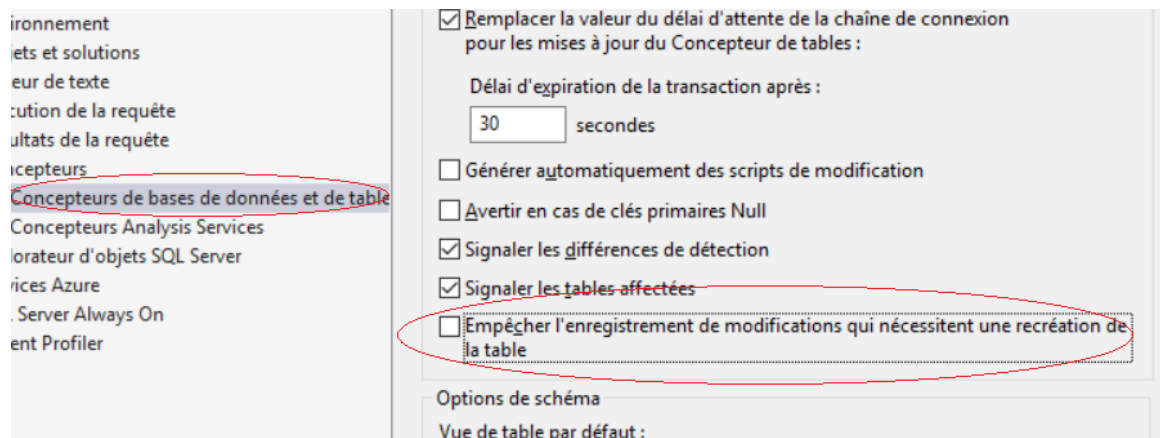
5. Faire OK, puis fermer pour terminer.
6. Enregistrez.

Attention : 

Si au moment d'enregistrer le diagramme vous avez cette fenêtre Alors



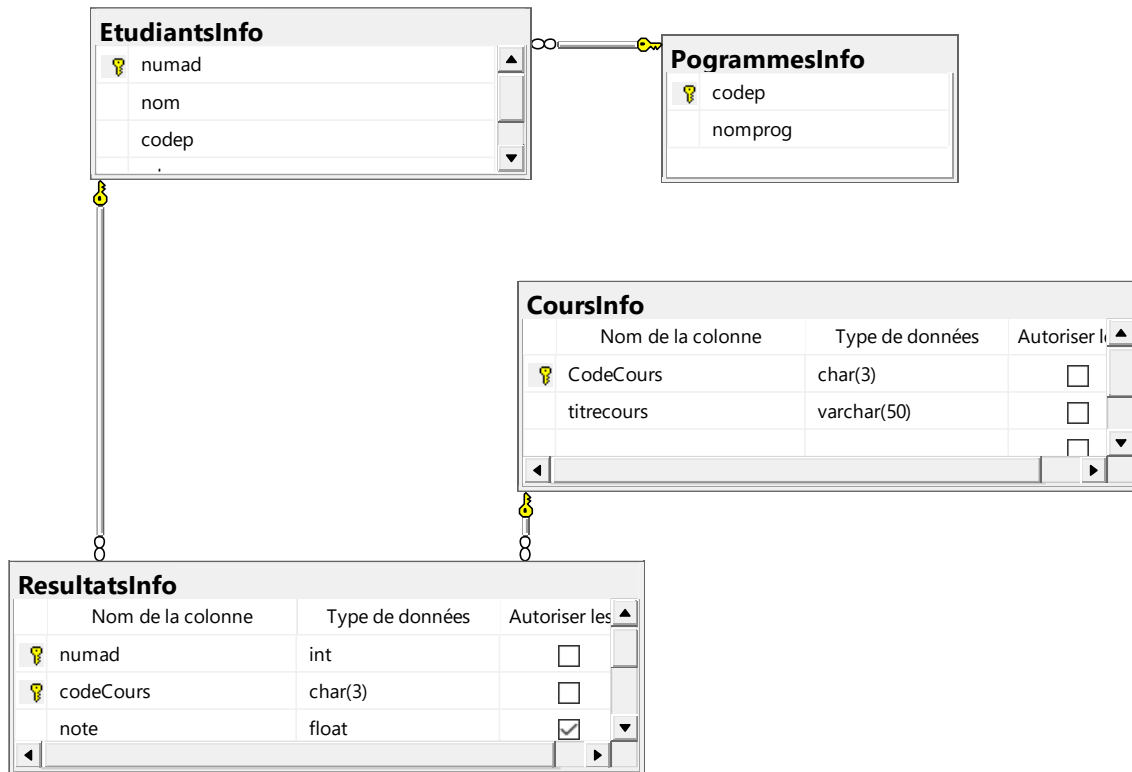
allez à Outils, puis Options, à concepteur de bases de données, décochez la case « Empêcher l'enregistrement » voir figure suivante



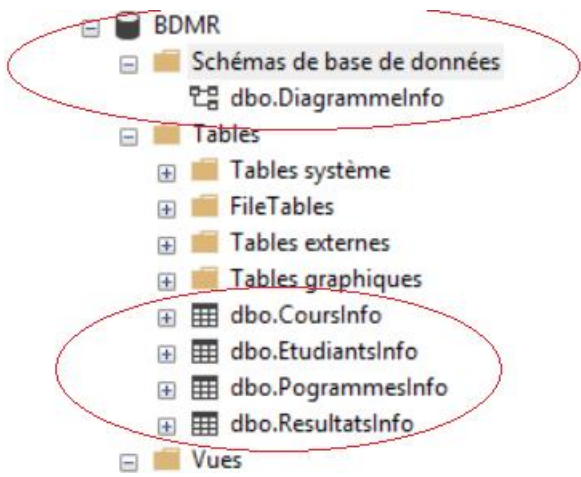
Définir la clé primaire composée

Pour définir une clé primaire composée sur une table, c'est très facile. Il suffit de sélectionner TOUTES les colonnes que l'on souhaite qu'elle soit clé primaire et d'ajouter une clé primaire comme au point 3 de l'étape 1. Il est probable que, les colonnes de

vosre clé primaire composées soient des clés étrangères comme dans la plupart des cas. Il faudra alors les définir comme telle.



Il n'est pas nécessaire de générer le code SQL pour créer les tables, puis que celles-ci sont déjà créées



Chapitre 5, éléments du langage

Transact-SQL

Définitions

La plupart des SGBDs relationnels offrent une extension du SQL, en y ajoutant des déclarations de variables, des structures de contrôles (alternatives et les répétitives) pour améliorer leurs performances

Transact-SQL ou T-SQL ou TSQL est l'extension du langage SQL pour Microsoft SQL Server et Sybase. Transact-SQL est un langage procédural permettant d'implémenter des fonctionnalités de bases de données que SQL seul ne peut implémenter.

Éléments du langage Transact-SQL :

Les variables et leurs déclarations

- Dans Transact SQL, on utilise le mot réservé DECLARE pour déclarer des variables.
- Les noms de variables sont précédés du symbole @
- Les types de variables, sont les types SQL
- Les variables peuvent être initialisées avec des valeurs en utilisant la fonction SET.

Exemple :

```
DECLARE  
@CHOIX int ;  
SET @CHOIX =1;
```

Les mots réservés : BEGIN ...END

Ces mots réservés permettent de définir un bloc ou un groupe d'instructions qui doivent être exécutées.

Les structures de contrôles

L'alternative :

L' Instruction IF

```
IF Boolean_expression  
    { sql_statement | statement_block }  
[ ELSE  
    { sql_statement | statement_block } ]
```

Ou encore

```
IF Boolean_expression
    { sql_statement | statement_block }
[ ELSE IF Boolean_expression
    { sql_statement | statement_block } ]
[ ELSE
    { sql_statement | statement_block } ]
```

Exemple: (ce bout de code est ce que nous appelons un bloc anonyme)

```
DECLARE
@sal money;
set @sal =40.00;
if @sal = (select salaire from etudiants where numad =8)
    (select * from etudiants where numad =8);
else if @sal=1.33 (select * from ETUDIANTS WHERE NUMAD=6);
else (select * from etudiants);
```

Attention: 

Lorsque vous avez un bloc d'instructions, celui-ci doit être placé entre BEGIN et END.

Exemple1:

```
DECLARE
@code char(3);
begin
set @code = 'tge';
if @code like '%'+ (select codep from etudiants where numad =6) + '%'
    (select * from etudiants where numad =6);
else (select * from ETUDIANTS WHERE NUMAD=1);
end;
```

Dans les exemples précédents, remarquez:

- Le bloc d'instructions BEGIN .. END
- Le IF ..ELSE
- Le IF ..ELSE IF ..ELSE
- Comment est construit le LIKE

Exemple2

```
DECLARE
@code char(3);
begin
set @code = 'tge';
    if @code like '%' + (select codep from etudiants where numad =6) + '%'
        (select * from etudiants where numad =6);
    else
    BEGIN
    (select * from ETUDIANTS WHERE NUMAD=1);
    INSERT INTO ETUDIANTS (NOM,PRENOM,SALAIRE,CODEP)
    VALUES('Mosus','Chat',12,'sim');
    update etudiants set salaire = salaire + 5 where codep = 'inf';
    END;
end;
```

Remarquez:

Après le ELSE, nous avons trois instructions à exécuter. Un SELECT, un INSERT et un UPDATE. Ces instructions sont placées entre BEGIN et END :

L'instruction CASE

Syntaxe :

```
CASE input_expression
    WHEN when_expression THEN result_expression [ ..n ]
    [ ELSE else_result_expression ]
END
```

Ou bien.

```
CASE
    WHEN Boolean_expression THEN result_expression [ ..n ]
    [ ELSE else_result_expression ]
END
```


Exemple 1, case avec un SELECT

```
SELECT nom, prenom, codep =  
CASE codep  
WHEN 'inf' THEN 'Informatique'  
WHEN 'tge' THEN 'Genie Ele'  
WHEN 'ele' THEN 'Electronique'  
ELSE 'Aucun Programme'  
END, salaire  
FROM etudiants ;
```

Exemple 2, de CASE dans un UPDATE

```
UPDATE etudiants  
SET salaire=  
( CASE  
WHEN (salaire < 5) THEN salaire + 40  
ELSE (salaire + 20.00)  
END  
) ;
```

La répétitive

La répétitive est implémentée à l'aide de la boucle WHILE.

Syntaxe :

```
WHILE Boolean_expression  
{ sql_statement | statement_block | BREAK | CONTINUE }
```

Exemple

Augmenter le salaire des étudiants, tant que la moyenne est inférieure à 80. Mais si le maximum des salaires dépasse 100 on arrête,

```
BEGIN  
WHILE (select avg(salaire) from etudiants )<= 80  
BEGIN update etudiants set salaire = salaire +10;  
IF(select max(salaire) from etudiants) >100 BREAK;  
ELSE CONTINUE;  
END;  
END;
```

Les curseurs :

Les curseurs sont des zones mémoire (mémoire tampon) utilisées par les SGBDs pour récupérer un ensemble de résultats issu d'une requête SELECT.

Pour MS SQL Server, les curseurs sont explicites, ce qui veut dire qu'ils sont associés à une requête SELECT bien précise. Comme par exemple, le curseur CUR1 contiendra le résultat de la requête : SELECT ename, job from emp where deptn=30;

Pour utiliser un curseur, nous avons besoin de le déclarer.

```
DECLARE nomCurseur CURSOR FOR SELECT ... FROM
```

Exemple :

```
DECLARE  
cur1 CURSOR FOR  
SELECT idcircuit, coutcircuit FROM circuits;
```

Pour lire le contenu d'un curseur, on procède comme suit :

- 1- Ouvrir le curseur avec **OPEN**.
- 2- Lire le curseur avec **FETCHINTO** et une boucle WHILE: pour aller chercher chaque enregistrement dans l'ensemble actif, une ligne à la fois, nous utiliserons la commande FETCH. À chaque fois que le FETCH est utilisé, le curseur avance au prochain enregistrement dans l'ensemble actif
- 3- Fermer le curseur avec la commande avec **CLOSE**
- 4- Supprimer la référence au Curseur avec **DEALLOCATE**

La fonction : **@@FETCH_STATUS** : Renvoie l'état de la dernière instruction FETCH effectuée sur un curseur. Elle renvoie 0 si tout s'est bien passé, -1 s'il n'y a plus de lignes, -2 si la ligne est manquante et -9 le curseur ne fait aucune opération d'extraction.

La fonction **@@CURSOR_ROWS**, renvoie le nombre de lignes qualifiantes actuellement dans le dernier curseur ouvert sur la connexion.

Exemple1

```

DECLARE @id int, @cout int;
DECLARE cur1 CURSOR FOR SELECT idcircuit, coutcircuit
FROM circuits;
BEGIN
OPEN cur1 ;
print concat('numero', '---', 'cout');

-- on initialise les variable @id et @cout avec le premier
FETCH(la première ligne)
FETCH NEXT FROM cur1 INTO @id, @cout;
-- Tant que le FETCH se fait normalement
WHILE @@FETCH_STATUS = 0
    BEGIN
        PRINT concat(@id, '-----', @cout);
        FETCH NEXT FROM cur1 INTO @id, @cout;
    END;
CLOSE cur1;
DEALLOCATE cur1;
END

```

Exemple 2

```

DECLARE @cout int;
DECLARE Cout_cursor CURSOR FOR SELECT coutcircuit FROM circuits;

BEGIN
OPEN Cout_cursor ;
FETCH NEXT FROM Cout_cursor INTO @cout;

WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @cout < 500 update circuits set coutcircuit = coutcircuit+
(coutcircuit*0.1) WHERE CURRENT OF Cout_cursor ;

        ELSE IF @cout BETWEEN 500 and 900 update circuits set coutcircuit
=coutcircuit+(coutcircuit*0.05) WHERE CURRENT OF Cout_cursor;

        ELSE update circuits set coutcircuit
=coutcircuit+(coutcircuit*0.01) WHERE CURRENT OF Cout_cursor;

        FETCH NEXT FROM Cout_cursor INTO @cout;
    END;
CLOSE Cout_cursor;
DEALLOCATE Cout_cursor;
END

```

Par défaut, les curseurs sont Forward ONLY : ils ne sont pas scrollables.
Lorsqu'un curseur est déclaré avec l'attribut SCROLL alors on peut accéder au contenu du curseur par d'autres option de la fonction FETCH. Nous pouvons avoir accès à la première ligne, la dernière ligne, une position absolue, exemple la ligne 3. Position relative à partir d'une position prédéfinie.

```
DECLARE Curmonument SCROLL CURSOR FOR
SELECT nomMonument , nbEtoiles FROM Monuments
ORDER BY nbEtoiles desc;

declare @nom varchar(30), @nb int;
BEGIN
OPEN Curmonument;

print(' la premiere ligne');
FETCH FIRST FROM Curmonument into @nom,@nb;
print concat(@nom, '----', @nb)

print('la dernière ligne');
FETCH LAST FROM Curmonument into @nom,@nb;
print concat(@nom, '----', @nb)

print('la ligne numero 3');
FETCH ABSOLUTE 3 FROM Curmonument into @nom,@nb;
print concat(@nom, '----', @nb)

print('la deuxième ligne après la ligne 3');
FETCH RELATIVE 2 FROM Curmonument into @nom,@nb;
print concat(@nom, '----', @nb)

print('le num immédiatement avant la position courante');
FETCH PRIOR FROM Curmonument into @nom,@nb;
print concat(@nom, '----', @nb)

print('le num qui est deux lignes avant la ligne courante');
FETCH RELATIVE -2 FROM Curmonument into @nom,@nb;
print concat(@nom, '----', @nb)

CLOSE Curmonument;
DEALLOCATE Curmonument;
END
```

Remarque : Nous reviendrons sur les détails concernant les curseurs plus loin dans le cours.

Chapitre 6, les procédures stockées

Définition

Une procédure stockée est un ensemble d'instructions SQL précompilées stockées dans le serveur de bases de données

Avantages à utiliser les procédures stockées

Il existe plusieurs avantages à utiliser des procédures stockées à la place de simple requêtes SQL

- Rapidité d'exécution, puisque les procédures stockées sont déjà compilées.
- Clarté du code : dans un code C#, PHP ou autre, il vaut mieux utiliser l'appel d'une procédure que l'instruction SQL, en particulier lorsque l'instruction SQL est longue et complexe.
- Faciliter le débogage.
- Réutilisation de la procédure stockée.
- Possibilité d'exécuter un ensemble de requêtes SQL
- Prévention d'injections SQL
- Modularité. Facilite le travail d'équipe.

Syntaxe simplifiée de définition d'une procédure stockée avec Transact-SQL

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }  
    [schema_name.] procedure_name  
    [ { @parameter data_type }  
      [ OUT | OUTPUT ] ]  
AS  
{ [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }  
[;]
```

CREATE PROCEDURE : indique que l'on veut créer une procédure stockée.

OR ALTER est optionnel, indique que l'on veut modifier la procédure stockée si celle-ci existe déjà.

@parameter data_type : On doit fournir la liste des paramètres de la procédure avec le type de données correspondant à chacun des paramètres.


[OUT | OUTPUT] : Indique la direction en OUT ou en OUTPUT des paramètres de la procédure. Par défaut les paramètres sont en IN. Lorsque les paramètres sont en IN, il n'est pas nécessaire (c'est même une erreur) d'indiquer la direction.

AS : mot réservé qui annonce le début du corps de la procédure et la fin de la déclaration des paramètres

BEGIN

Bloc SQL ou Transact-SQL

END;

Attention : 

Les paramètres sont précédés du symboles @

Le type de paramètre IN OUT est indiqué uniquement si le paramètre est en OUT ou INOUT (le type IN est par défaut) : La direction IN provoque une erreur si indiquée.

Exemple1 : Tous les paramètres sont en IN. (Insertion)

```
create procedure insertionEtudiants
(
@pnom varchar(20), @pprenom varchar(30),@psal
money,@pcodep char(3)
)
AS
begin
insert into etudiants(nom , prenom ,salaire ,codep )
values (@pnom , @pprenom ,@psal ,@pcodep)
end;
```

Exécution d'une procédure dans son SGBD natif (MS SQL Server)

Pour exécuter une procédure stockée, on utilise les commandes execute ou exec. Il faudra fournir la valeur des paramètres.

Exemple :

```
execute insertionEtudiants
@pnom = 'Lenouveau',
@pprenom = 'lenouveau',
@psal=22.5,
@pcodep = 'sim';
```

Même s'il est conseillé de passer les paramètres dans l'ordre de leur apparition dans la procédure, MS SQL Server peut accepter la passation des paramètres dans n'importe quel ordre. Par contre, les noms des paramètres sont très importants. En ce sens SQL Server est contraire d'ORACLE (pour ORACLE c'est l'ordre des paramètres qui est important et non le nom)

On aurait très bien pu faire ceci , le paramètre @nom est fourni en dernier.

```
execute insertionEtudiants  
@pprenom = 'aaaa',  
@psal=22.5,  
@pcodep = 'sim',  
@pnom = 'patate'
```

 Exemple 2 : Les paramètres en IN avec une sortie (SELECT)

```
create procedure lister  
(  
@pcodep char(3)  
)  
AS  
begin  
select nom,prenom from etudiants where @pcodep = codep;  
end;
```

Execution:

```
execute lister  
@pcodep='inf';
```

Exemple 3, utilisation de LIKE dans une procédure stockée

```
create procedure ChercherNom  
(  
@pnom varchar(20)  
)  
AS  
begin
```

```
select * from etudiants where nom Like '%' + @pnom + '%';  
end;
```

Execution

```
execute ChercherNom  
@pnom='Le';
```

Exemple 4 : Procédure avec un paramètre en OUTPUT

```
create procedure ChercherNom2  
(  
@pnum int,  
@pnom varchar(20) out  
)  
AS  
begin  
  
select @pnom = nom  
from etudiants where numad =@pnum;  
end;  
go
```

Execution

```
declare @pnum int =1;  
declare @pnom varchar(20);  
execute ChercherNom2  
@pnum ,  
@pnom output;  
print @pnom;
```


Les fonctions stockées : Syntaxe simplifiée.

Les fonctions stockées sont des procédures stockées qui retournent des valeurs. Leurs définitions sont légèrement différentes d'une procédure stockée mais le principe général de définition reste le même.

Cas d'une fonction qui ne retourne pas une table

```
CREATE [ OR ALTER ] FUNCTION [ schema_name. ] function_name
( [ { @parameter_name parameter_data_type
    } ]
)
RETURNS return_data_type


[ AS ]
BEGIN
    function_body
    RETURN scalar_expression
END
[ ; ]
```

Exemple 1, fonction avec paramètres

```
create function compteretudiants(@pcode char(3)) returns int
as
begin
declare @total int;
select @total = count(*) from Etudiants where codep =@pcode;
return @total;
end
go
```

Exécution d'une fonction dans MS SQL Server

Pour exécuter une fonction qui ne retourne pas une table, il faudra utiliser la commande **SELECT**, suivie du nom de la fonction Il faudra passer les valeurs des paramètres pour la fonction.

Attention : 

1. Pour l'appel des fonction (Eexécution), nous avons besoin de préciser le shéma de la BD. Le shéma est toujours : nomUtilisateur.nomObjet .
2. Pour l'instant, tous les objets appartient à l'usager **dbo**.
3. Pour une fonction qui ne retourne pas une table, pas besoin du FROM pour le select.

Remarque : le mappage des utilisateurs aux connexions, sera abordé plus loin.

Pour exécuter la fonction précédente :

```
select dbo.compteretudiants('inf');  
---Pas de clause FROM.
```

Exemple2 : fonction sans paramètres

```
create function compter() returns int  
as  
begin  
declare @total int;  
select @total = count(*) from etudiants;  
return @total;  
end;
```

```
select dbo.compter();  
--pas de clause FROM
```

Cas d'une fonction qui retourne une table.

```
CREATE [ OR ALTER ] FUNCTION [ schema_name. ] function_name  
( [ { @parameter_name parameter_data_type  
      } ]  
)  
RETURNS TABLE  
  
[ AS ]  
RETURN [ ( ) select_stmt [ ) ]  
[ ; ]
```

Exemple

```
Create FUNCTION Cherchertousetudiants  
(@pcodep char(3)) returns table  
AS  
return(  
SELECT nom, prenom  
FROM etudiants  
WHERE @pcodep = codep  
);  
GO
```

L'appel (Exécution) d'une fonction qui retourne une table est différent. Le SELECT dans ce cas, doit utiliser la clause FROM puis que ce qui est retourner est une table. De plus, si la fonction a des paramètres en IN (implicite) il faudra les déclarer et leur affecter des valeurs.

```
declare @codep char(3);  
set @codep='inf';  
select * from Cherchertousetudiants(@codep);
```

Supprimer une fonction ou une procédure :

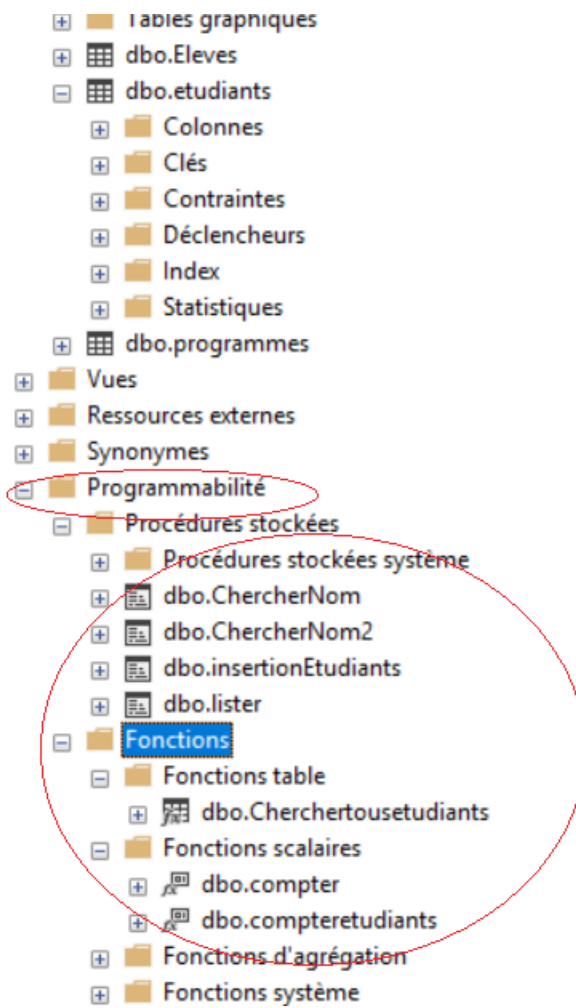
Les fonctions et les procédures sont des objets de la base de données. Ils se détruisent donc avec la commande DROP

```
drop procedure ChercherNom2;
```

```
drop function compteretudiants
```

En conclusion pour les procédures et les fonctions.

- Pour les procédures et les fonctions les paramètres sont précédés de @
- Le type IN est par défaut.
- Lorsque le paramètre est en OUT ou OUTPUT, il faudra l'indiquer clairement.
- Les procédures et fonctions sont terminées par GO. Il n'est cependant pas obligatoire.
- Le mot réservé DECLARE est obligatoire pour déclarer des variables.
- Les fonctions peuvent retourner des tables. Elles ne comportent pas les mots réservés BEGIN et END
- Pour exécuter une procédure il faut utiliser **execute** ou **exec**
- Pour exécuter une fonction il faut utiliser **select** nomuser.nomfonction (valeur paramètres)
- À l'exécution des procédures, l'affectation des valeurs aux paramètres se fait avec = pour les int et la fonction **set** pour les types text.
- Vos fonctions et procédures se trouvent à Programmabilité de la BD



Les procédures stockées et les fonctions : les Templates.

Voici le code généré par SQL Server lorsque vous essayer de créer une procédure ou une fonction

```
=====
-- Template generated from Template Explorer using:
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-- =====
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =====
-- Author:          <Author,,Name>
-- Create date:    <Create Date,,>
-- Description:    <Description,,>
-- =====
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
    -- Add the parameters for the stored procedure here
    <@Param1, sysname, @p1> <Datatype_For_Param1, , int> =
<Default_Value_For_Param1, , 0>,
    <@Param2, sysname, @p2> <Datatype_For_Param2, , int> =
<Default_Value_For_Param2, , 0>
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
GO
```

Template function TABLE

```
-- =====
-- Template generated from Template Explorer using:
-- Create Inline Function (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the function.
-- =====
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =====
-- Author:          <Author,,Name>
-- Create date:    <Create Date,,>
-- Description:    <Description,,>
-- =====
CREATE FUNCTION <Inline_Function_Name, sysname, FunctionName>
(
    -- Add the parameters for the function here
    <@param1, sysname, @p1> <Data_Type_For_Param1, , int>,
    <@param2, sysname, @p2> <Data_Type_For_Param2, , char>
)
RETURNS TABLE
AS
RETURN
(
    -- Add the SELECT statement with parameter references here
    SELECT 0
)
GO
```

Chapitre 7, les Triggers ou déclencheurs

Définition :

Les triggers sont des procédures stockées qui s'exécutent automatiquement quand un événement se produit. En général cet événement représente une opération DML (Data Manipulation Language) sur une table. Les instructions DML doivent inclure INSERT, UPDATE ou DELETE

Rôle des triggers :

- Contrôler les accès à la base de données
- Assurer l'intégrité des données
- Garantir l'intégrité référentielle (DELETE, ou UPDATE CASCADE)
- Tenir un journal des logs.

Même si les triggers jouent un rôle important pour une base de données, il n'est pas conseillé d'en créer trop. Certains triggers peuvent rentrer en conflit, ce qui rend l'utilisation des tables impossible pour les mises à jour.

Syntaxe simplifiée pour créer un trigger avec une opération DML

Syntaxe simplifiée :

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_statement }
```

AFTER spécifie que le déclencheur DML est déclenché uniquement lorsque toutes les opérations spécifiées dans l'instruction SQL ont été exécutées avec succès.

Un trigger utilisant AFTER va effectuer l'opération DML même si celle-ci n'est pas valide, un message erreur est quand même envoyé.

Utilisez les triggers AFTER avec une ROLLBACK TRANSACTION

Le FOR fait la même chose que AFTER, donc il va quand même insérer ou mettre à jour. Par défaut on utilise AFTER.

INSTEAD OF indique un ensemble d'instructions SQL à exécuter à la place des instructions SQL qui déclenche le trigger.

Au maximum, un déclencheur INSTEAD OF par instruction INSERT, UPDATE ou DELETE peut être défini sur une table ou une vue. → Définir des vues pour des vues pour des INSTEAD OF.

PAS de INSTEAD OF sur des vues avec l'option with CHECK OPTION.

Pour INSTEAD OF pas d'instruction DELETE sur des tables ayant l'option ON DELETE CASCADE (idem pour UPDATE)

[Principe de fonctionnement pour les triggers DML.](#)

Lors de l'ajout d'un enregistrement pour un Trigger ...INSERT, le SGBD prévoit de récupérer l'information qui a été manipulée par l'utilisateur et qui a déclenché le trigger. Cette information (INSERT) est stockée dans une table temporaire appelée **INSERTED**.

Lors de la suppression d'un enregistrement, DELETE, le SGBD fait la même chose en stockant l'information qui a déclenché le trigger dans une table temporaire appelée **DELETED**.

Lors, d'une mise à jour, UPDATE l'ancienne valeur est stockée dans la table DELETED et la nouvelle valeur dans INSERTED.

[Exemple 1, suppression en cascade](#)

```
create trigger deletescascade on departements
instead of delete as
begin
declare
@code char(3);
    SELECT @code = deptno FROM deleted;
    delete from EmpPermanent where deptno =@code;
    delete from Departements where deptno=@code;
end;
```


Exemple 2

```
create TRIGGER ctrlSalairePermanent on EmpPermanent after update
as
declare
@ancienne money,
@nouvelle money;

BEGIN
select @ancienne = Salaire from deleted ;
select @nouvelle = Salaire from inserted;
IF (@ancienne > @nouvelle)
    rollback;
    RAISERROR (15600,-1,-1, 'pas bon salaire');
END;
```

Exemple 3

Le contenu de la table Emplois

typeEmplois	salaireMin	salaireMax
Analystes	75000,00	140000,00
Directeur	80000,00	200000,00
Finances	45000,00	120000,00
Programmeurs	55000,00	100000,00

Le contenu de la table EmployesBidon

	empno	nom	pre nom	salaire	typeEmplois
1	1	Patoche	Alain	100000,00	Directeur
2	2	Leroy	Alain	55000,00	Finances
3	4	Lemieux	Thierry	80000,00	Programmeurs

Le trigger ci-dessous fait en sorte que les salaires des employés respectent la fourchette des salaires définie dans la table Emplois.

```

CREATE TRIGGER CTRLSALAIRES on employesBidon
after INSERT, UPDATE as

DECLARE
@minsalaire money,
@maxsalaire money,
@newsalaire money;
BEGIN
SELECT @minsalaire = salaireMin from emplois WHERE typeemploi =
(select typeemploi from inserted);

SELECT @maxsalaire =salaireMax from emplois WHERE typeemploi =
(select typeemploi from inserted);

select @newsalaire = salaire from inserted;

if (@newsalaire<@minsalaire or @newsalaire>@maxsalaire)

rollback TRANSACTION;

else commit transaction;
end;

```

RAISERROR:

Génère un message erreur défini par l'utilisateur. Le message n'arrête pas le trigger (ce n'est pas comme Raise_Application_error d'Oracle).

```
RAISERROR(id_message, sévérité ,État , 'Message');
```

id_message, indique le numéro du message. Ce numéro doit être >50000. Lorsqu'il n'est pas indiqué ce numéro vaut 5000.

Sévérité : indique le degré de gravité associé au trigger, ce niveau de gravité est défini par l'utilisateurs. Ce nombre se situe entre 0 et 25. Les utilisateurs ne peuvent donner que le nombre entre **0 et 18**. Les nombre entre 19 et 25 sont réservés aux membres du groupe sysadmin. Les nombre de 20 à 25 sont considérés comme fatals. Il est même possible que la connexion à la BD soit interrompue.

Si ce nombre est négatif, il est ramené à 1.

Exemples :

Erreur :	Sévérité
Duplication de Clé primaire	14
Problème de FK	16
Problème insertion (valeurs non conformes)	16
Violation de contrainte Check	16
Trigger DML	15 ou 16

Si vous prêtez attention aux messages erreurs renvoyés par le SGBD, vous constaterez qu'ils se présentent sous la forme du RAISERROR vous pouvez vous baser sur ces messages pour fixer le degré de sévérité.

État : utilisé lorsque la même erreur définie par l'utilisateur se retrouve à plusieurs endroits, l'état qui est un numéro unique permet de retrouver la section du code ayant générée l'erreur. L'état est un nombre entre 0 et 255. Les valeurs >255 ne sont pas utilisées. Si négatifs alors ramenés à 0.

Exemples :

```
insert into EmpPermanent values(88,41111,12,'inf');
```

Ici, nous avons un problème de Foreign key puisque le 88 n'est pas un dans la table EmpClg. Pour la première fois, le niveau de sévérité est 16 est l'état est 0.

Vous pouvez également utiliser un try ---catch pour récupérer le message erreur proprement : Dans le cas de l'exemple 2

```

use EmpcIlgDB;
begin try
    begin transaction;
        update EmpPermanent set Salaire =1 where empno =12;
        commit transaction;
end try

begin catch
    select ERROR_MESSAGE() as message, ERROR_SEVERITY() as Gravit ,
        ERROR_STATE() as etat, @@TRANCOUNT
        if @@TRANCOUNT>0 rollback;
end catch;

```

	message	Gravit�	etat
1	An invalid parameter or option was specified for procedure 'Le salaire ne doit pas �tre r�vis� � la baisse'.	15	1

Message : repr sente le message d fini par l'utilisateur. Au maximum 2047 caract res.
 Vous pouvez  galement laisser le soin au SGBDR d'utiliser ses propres param tres.

Attention : 

Les triggers sont d finis sur une table , ce sont donc des objets de la table, tout comme une colonne, une contrainte...

Activer /d sactiver un trigger

Utiliser la commande DISABLE pour d sactiver temporairement un trigger

```

DISABLE TRIGGER { [ schema_name . ] trigger_name [ ,...n ] | ALL }
ON { object_name | DATABASE | ALL SERVER } [ ; ]

```

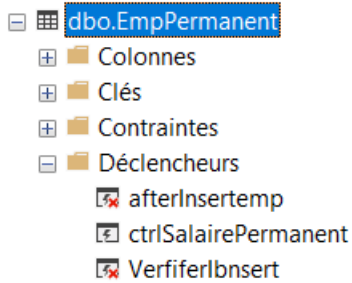
Exemples :

```

disable trigger [dbo].[afterInsertemp] ,[dbo].[VerfiferInsert]
on [dbo].[EmpPermanent];

```

Un trigger d sactiv  va toujours exister dans le syst me mais ne fait rien. (sans action).
 Dans Management Studio, il est marqu  **en rouge**.



Pour réactiver votre trigger, utiliser la commande ENABLE. Cette commande a la même syntaxe que la commande DISABLE.

```
Enable trigger [dbo].[VerfiferIbnsert] on [dbo].[EmpPermanent];
```

Supprimer un trigger.

Un trigger est un objet de la base de données, il faudra utiliser la commande DROP pour le détruire.

```
DROP TRIGGER [ F EXISTS ] [schema_name.]trigger_name [ ,.n ] [; ]
```

Exemple :

```
DROP TRIGGER [dbo].[VerfiferInsert];
```

Retour sur la commande CREATE TABLE : ON DELETE CASCADE

Les triggers sont un bon moyen de contrôler l'intégrité référentielle (→ la Foreign KEY) lors de la suppression d'un enregistrement référencé (ou des enregistrements référencés). Si lors de votre conception, vous avez déterminé que les enregistrements liés par la Foreign KEY doivent être supprimés car il s'agit d'un lien de composition, comme dans le cas d'un livre et ses chapitres, c'est-à-dire que lorsqu'un livre est supprimé alors tous les chapitres liés à ce livre doivent être également supprimé, ou encore lorsqu'il s'agit d'une relation de généralisation, alors vous pouvez le faire à la création de table.

Exemple

Voici la création de la table livres

```

create table livres
(
coteLivre char(5),
titre varchar(40) not null,
langue varchar(20) not null,
annee smallint not null,
nbPages smallint not null,
constraint pklivre primary key(coteLivre)
);

```

Voici la table Chapitres

```

create table Chapitres
(
idChapitre char(7) constraint pkChapitre primary key,
nomChapitre varchar(40) not null,
coteLivre char(5) not null,
constraint fkLivre foreign key (coteLivre)
references livres(coteLivre)ON DELETE CASCADE
)

```

Lorsqu'un livre (ou des livres) sont supprimés alors les chapitres de ce livre le sont aussi.

Attention : 

La suppression en cascade à la création des tables n'est pas toujours recommandée sauf si la conception l'exige....

Pour tester :

```

---insertion dans livres--
begin transaction trans1
insert into livres values('IF001', 'Introduction à C#', 'Français', 2017, 650);
insert into livres values('IF002', 'SQL pour Oracle 12C', 'Français', 2015, 500);
insert into livres values('IF003', 'Oracle pour Java et PHP', 'Français', 2016, 700);
insert into livres values('IF004', 'Windows Server 2016', 'Anglais', 2016, 1100);
insert into livres values('MA001', 'Algèbre Linéaire', 'Français', 2013, 400);
commit transaction trans1;

---insertion dans Chapitres
begin transaction trans2
insert into Chapitres values('IF00101', 'Pour bien commencer ', 'IF001');
insert into Chapitres values('IF00102', 'introduction à la POO ..', 'IF001');

```

```

insert into Chapitres values('IF00110','les tableaux ','IF001');
insert into Chapitres values('IF00201','Concepts de bases de données ','IF002');
insert into Chapitres values('IF00202','Create table ..','IF002');
insert into Chapitres values('IF00212','les indexs','IF002');
insert into Chapitres values('MA00101','introduction ','MA001');
insert into Chapitres values('MA00102','Les vecteurs','MA001');
insert into Chapitres values('MA0013','les matrices','MA001');
commit transaction trans2;

--pour tester
---en 1
begin transaction trans3;
delete from livres where coteLivre ='MA001' or coteLivre = 'IF002';
---en 2
rollback transaction trans3;

```

Maintenant, si votre conception initiale, ne doit pas faire de suppression en cascade comme par exemple les employés et les départements, alors opter pour un trigger.

Exemple :

```

USE [Empc1gDB]
GO
/***** Object: Trigger [dbo].[deletescdcDepartement] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER trigger [dbo].[deletescdcDepartement] on
[dbo].[Departements]
instead of delete as
begin
declare
@code char(3);
SELECT @code = deptno FROM deleted;
update EmpPermanent set deptno = null where deptno =@code;
delete from Departements where deptno=@code;
end;

```

En conclusion :

1. Pour garantir l'intégrité de données en général et référentielle en particulier, faites-le par la base de données au CREATE TABLE. Comme les PK, le FK, Les Check...
2. Les triggers sont là pour renforcer l'intégrité des données. Leur avantage est qu'on peut les désactiver au besoin. De plus ils s'exécutent automatiquement (même s'ils sont oubliés).
3. Les procédures stockées sont un excellent moyen pour réduire les risques de briser l'intégrité des données, à condition qu'elles soient utilisées.
4. À moins que ce soit obligé, évitez le ON DELETE CASCADE.

Bonnes pratiques pour les procédures STOCKÉES :

1. Éviter les SELECT*
2. Utilisez des transactions explicites : BEGIN /COMMIT TRANSACTION et privilégiez des transactions courtes. Les transactions longues utilisent un verrouillage plus long.
3. À partir de MS SQL server 2016 vous avez la fonction TRY...CATCH, utilisez là si possible.

```
create procedure insertDept(@code char(3), @nom varchar(30)) as
begin
    begin try
        begin transaction;
        insert into deparatements values(@code,@nom);
        commit transaction;
    end try
    begin catch
        if(@@TRANCOUNT>0)
            rollback;
    end catch;
end;
```

4. Privilégiez des jointures à la place de sous-requêtes.
5. Restreindre les résultats le plus tôt possible. (WHERE). Éviter des fonctions qui retournent un trop gros nombre de données
6. Des procédures peuvent en appeler d'autres. Vous avez jusqu'à 32 niveaux d'imbrications. Mais faites attention

Chapitre 8, les transactions

Notions de Transactions :

Une transaction est un bloc d'instructions DML exécutés et qui laisse la base de données dans un état cohérent. Si une seule instruction dans le bloc n'est pas cohérente alors la transaction est annulée, toutes les opérations DML sont annulées. Le principe de transaction est implémenté dans tous les SGBDs.

Exemple :

```
begin transaction trans1;
insert into Departements values('dept', 'resources humaines');
update EmpPermanent set deptno = 'inf' where empno=1;
update EmpPermanent set Salaire = 45000 where empno =1;
insert into Departements values('dept', 'resources humaines');
commit transaction trans1;
```

Le bloc d'instruction précédent ne va s'exécuter puisque nous avons un problème avec le INSERT, la clé primaire est dupliquée.

Propriétés d'une transaction

Les transactions ont la propriété **ACID**

A : pour Atomicité :

Une transaction doit être une unité de travail indivisible ; soit toutes les modifications de données sont effectuées, soit aucune ne l'est.

C : pour la Cohérence

Lorsqu'elle est terminée, une transaction doit laisser les données dans un état cohérent. Dans une base de données relationnelle, toutes les règles doivent être appliquées aux modifications apportées par la transaction, afin de conserver l'intégrité de toutes les données.

Des fonctionnalités de gestion des transactions qui assurent l'atomicité et la cohérence des transactions. Lorsqu'une transaction a débuté, elle doit se dérouler correctement jusqu'à la fin (validée), sans quoi l'instance du Moteur de base de données annule toutes les modifications effectuées sur les données depuis le début de la transaction. Cette opération est appelée restauration d'une transaction, car elle retourne les données telles qu'elles étaient avant ces modifications.

I : pour Isolement

Les modifications effectuées par des transactions concurrentes doivent être isolées transaction par transaction. Une transaction reconnaît les données dans l'état où elles se trouvaient avant d'être modifiées par une transaction simultanée, ou les reconnaît une fois que la deuxième transaction est terminée, mais ne reconnaît jamais un état intermédiaire.

Des fonctionnalités de verrouillage (verrou ou LOCK) permettant d'assurer l'isolement des transactions.

D : Durabilité

Lorsqu'une transaction durable est terminée, ses effets sur le système sont permanents. Les modifications sont conservées même en cas de défaillance du système

Des fonctionnalités de consignation assurent la durabilité des transactions. Pour les transactions durables, l'enregistrement du journal est renforcé sur le disque avant les validations des transactions. Ainsi, en cas de défaillance du matériel serveur, du système d'exploitation ou de l'instance du Moteur de base de données lui-même, l'instance utilise au redémarrage les journaux des transactions pour restaurer automatiquement toutes les transactions incomplètes jusqu'au moment de la défaillance du système

Pour SQL SERVER certaines transactions sont atomique et donc auto-commit, instruction individuelle qui n'ont pas de BEGIN Transaction.

D'autres transaction sont explicites, dans ce cas elle commence par un : BEGIN TRANSACTION et se termine par un COMMIT Transaction ou un ROLLBACK.

BEGIN TRANSACTION : est comme un point, ou un état où les données référencées par une connexion sont **cohérentes logiquement et physiquement**. En cas d'erreur, toutes les modifications de données effectuées après BEGIN TRANSACTION peuvent être annulées pour ramener les données à cet état de cohérence connu. Chaque transaction dure jusqu'à ce qu'elle soit terminée proprement par un COMMIT ou par un ROLLBACK ;

À chaque BEGIN TRANSACTION, le système incrémente la variable @@TRANSCOUNT de 1. Cette variable système retourne le nombre de BEGIN Transaction exécuté pendant la

connexion en cours. Lorsqu'une transaction est comitée (COMMIT) alors

@@TRANCOUNT décrémente de 1. Le ROLLBACK TRANSACTION décrémente la variable **@@TRANCOUNT** jusqu'à 0. (La base de données est dans un état cohérent)

Récupération d'une transaction

Une transaction débute par un **begin transaction** et termine par un **commit** ou un **rollback**.

L'opération **commit** détermine le point où la base de données est de nouveau **cohérente**.

L'opération **rollback** annule toutes les opérations et retourne la base de données dans l'état où elle était au moment du **begin transaction**, donc du dernier **commit**.

Une transaction n'est pas uniquement une unité logique de traitement des données, c'est aussi une **unité de récupération**.

Après qu'une transaction ait terminé avec succès (commit) le SGBDR garantit que les changements seront permanents dans la BD. Cela ne veut pas dire, cependant, que les changements ont été écrits sur le disque dans le fichier physique de la BD. Ils peuvent être encore seulement dans la mémoire de l'ordinateur.

Supposons que 1 seconde après le commit et avant que les changements soient écrits sur le disque, une panne électrique vient tout effacer le contenu de la mémoire et en même temps les changements tout juste 'comités'.

Dans une telle situation, le SGBDR sera quand même capable, au redémarrage, de poursuivre la mise à jour en récupérant la transaction des journaux. Cela est possible à cause d'une règle qui stipule que les journaux sont physiquement sauvegardés sur le disque avant que le commit complète.

Cette double sauvegarde ou redondance des données permet de récupérer non seulement une transaction, mais une BD complète advenant une panne du disque.

Récupération complète de la base de données

Au moment d'une panne électrique ou d'une panne d'ordinateur, le contenu de la mémoire est perdu. L'état des transactions en cours est perdu. Les transactions complétées, mais non écrites sont disponibles dans les journaux.

Au moment du redémarrage du SGBDR, toutes les transactions qui n'ont pas complété seront annulées. Celles qui n'ont pas été sauvegardées dans la BD seront rejouées à partir des journaux.

À intervalle régulier le SGBDR sauvegarde le contenu de ses structures de données en mémoire dans le fichier physique de la BD. Au même moment un enregistrement 'CheckPoint' est ajouté au journal indiquant que toutes les transactions complétées avant le CP sont contenues dans la BD sur le disque.

Pour déterminer quelles transactions seront annulées et quelles transactions seront rejouées, le SGBDR utilise cet enregistrement CP dans le journal.

Supposons la situation suivante

Temps →	tcp		tp	
T1	[Green bar]			
T2		[Green bar]		
T3		[Red bar]	[Red bar]	
T4			[Green bar]	
T5			[Red bar]	
	Checkpoint		Panne	

Figure 1: États de 5 transactions au moment de la panne dans le journal des transactions

- Le SGBDR tombe en panne au temps **tp**;
- Le 'CheckPoint' le plus récent avant la panne est au temps **tcp**;
- T1 a complété avant tcp; donc sauvegardé dans le fichier;
- T2 a débuté avant tcp et a complété après, mais avant la panne à tp; donc pas écrit dans le fichier;
- T3 a débuté avant tcp mais n'a pas complété avant la panne à tp;
- T4 a débuté et complété après tcp; pas écrit dans le fichier;
- T5 a débuté après tcp mais n'a pas complété avant la panne à tp;

Transactions concurrentes

Un SGBDR permet à plusieurs transactions d'accéder la même information en même temps. Pour éviter que les transactions interfèrent l'une avec l'autre, des mécanismes sont nécessaires pour contrôler l'accès aux données.

Transaction A	temps	Transaction B
A(a = 40)		A(a = 40)
Update A set A.a = A.a - 20 where ...;	t1	
	t2	Update A set A.a = A.a - 5 where
Commit A(a = 20)		
		Commit A(a = 15)

Perte de mise à jour

Transaction A	temps	Transaction B
A(a = 40)		A(a = 40)
select @v = A.a from A where ...	t1	
@v = @v - 20 (@v == 20)		
	t2	select @v = A.a from A where ...
		@v = @v - 10 (@v = 30)
If @v <= 15 rollback		If @v <= 15 rollback
Update A set A.a = @v where ...;	t3	
	t4	Update A set A.a = @v where ...
Commit A(a = 20)	t5	
	t6	Commit A(a = 30) → A(a = 10)

Les verrous

Le verrouillage est un mécanisme utilisé par le Moteur de base de données SQL Server pour synchroniser l'accès simultané de plusieurs utilisateurs à la même donnée.

Avant qu'une transaction acquière une dépendance sur l'état actuel d'un élément de données, par exemple par sa lecture ou la modification d'une donnée, elle doit se protéger des effets d'une autre transaction qui modifie la même donnée. Pour ce faire, la transaction demande un verrou sur l'élément de données. Le verrou possède plusieurs modes, par exemple partagé ou exclusif. Le mode de verrouillage définit le niveau de dépendance de la transaction sur les données

Le tableau suivant illustre les modes de verrouillage des ressources utilisés par le Moteur de base de données.

Mode de verrouillage	Description
Partagé (S)	Utilisé pour les opérations de lecture qui n'effectuent aucune modification ou mise à jour des données, par exemple une instruction SELECT
Mise à jour (U)	Utilisé pour les ressources pouvant être mises à jour. Empêche une forme de blocage courante qui se produit lorsque plusieurs sessions lisent, verrouillent et mettent à jour des ressources ultérieurement.
Exclusif(X)	Utilisé par les opérations de modification de données, telles que INSERT, UPDATE ou DELETE. Empêche des mises à jour multiples sur la même ressource au même moment.

Chapitre 9, optimisation de requêtes

Introduction.

En principe, lorsqu'une requête SQL est envoyée au SGBD, celui-ci établit un plan d'exécution. Le module se charge d'établir un plan d'exécution s'appelle Optimizer.

Le fonctionnement de l'Optimizer globalement similaire pour l'ensemble des SGBDs (Oracle et SQL Server), en utilisant les étapes suivantes :

1. Validation syntaxique
2. Validation sémantique
3. Utilisation éventuelle d'un plan précédemment produit
4. Réécriture/Simplification de la requête
5. Exploration des chemins d'accès et estimation des coûts.
6. Désignation du chemin le moins coûteux, génération du plan d'exécution et mise en cache de ce dernier.

Les index


Un index est un objet de la base de données permettant d'accélérer l'accès aux données. C'est un peu comme un code postal qui permet à un facteur de retrouver une adresse rapidement ou comme une recherche de livres dans une bibliothèque ou alors comme une recherche d'information dans un livre, voir la table d'index à la fin. Le principe est d'aller directement à l'information souhaitée dans le cas d'un livre plutôt que de lire le livre au complet de manière séquentielle pour trouver l'information recherchée.

Le principe de recherche dans un index se fait un peu comme dans un B-Arbre un arbre parfaitement équilibré.

Le rôle d'un index est d'accélérer la recherche d'information (lors d'un SELECT) dans une base de données.

Par défaut, TOUS les SGBD entretiennent un index primaire qui est l'index créé sur la clé primaire. Cependant les développeurs peuvent décider de créer d'autres index sur des colonnes qui ne sont pas des PK.

- Créer des index sur les colonnes de Foreign KEY pour accélérer les jointures, sauf si la combinaison de FK forme une clé primaire (redondance d'index).
- Créer des index sur les colonnes de la clause WHERE sauf si le WHERE contient un like de fin (WHERE nom like '%CHE'), ou si le WHERE contient une fonction.
- Créer des index sur des colonnes utilisées dans un ORDER BY, un GROUP BY, un HAVING.
- Créer des index sur une colonne ayant une petite plage de valeurs inutiles. (NULL)
- Créer des index une fois que les insertions sont complétées.

Attention : 

Même si les index sont des accélérateurs, trop d'index ralentit le SGBD. Il ne faudrait pas que le SGBD passe son temps à maintenir TOUS les index. Les index ralentissent le système durant les insertions, car la table des index doit être mise à jour.

Types d'index :

MS SQL server manipule deux types d'index : CLUSTERED index et les NON CLUSTERED index

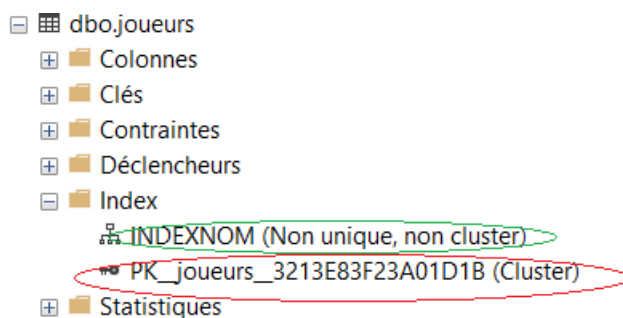
Les CLUSTERED INDEX :

Il existe un seul CLUSTERED index par table. Ces index stockent les lignes de données de la table en fonction de leurs valeurs de clé. Les index clustérisés trient et stockent les lignes de données dans la table ou la vue en fonction de leurs valeurs de clé

En principe, toutes les tables devraient avoir un index cluster défini sur la ou les colonnes ayant la propriété d'unicité ou de clé primaire. Par défaut lorsque SQL server crée une table avec clé primaire, il y ajoute un CLUSTERED index .

Exemple, remarquez la table joueurs suivantes créé avec un index Cluster sur la clé primaire. Cet index est créé à la création de la table joueurs dès que la clé primaire a été indiquée.

L'index non cluster (vert) a été rajouté par le développeur



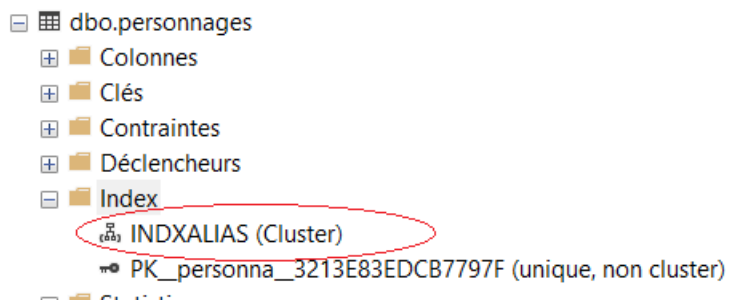
Si vous voulez mettre un autre index Cluster sur votre table il faudra :

1. À la création de table indique que la PK n'est pas un index Cluster

```
create table personnages(id int identity(1,1) not null primary
key nonclustered,
alias varchar(10) NOT NULL,
nom varchar(30) not null,
descriptions varchar(60) not null,
typ char(1) not null
);
```

2. Créer un Index Cluster sur la colonne que vous souhaitez.

```
CREATE CLUSTERED INDEX INDXALIAS ON personnages (ALIAS);
```



Sur quelles colonnes est-ce qu'il est conseillé de créer des index Clustérisés ?

- Des colonnes avec des valeurs uniques ou très peu de valeurs identiques.
- Colonne définie avec IDENTITY
- Colonnes fréquemment utilisées pour trier (ORDER BY) les données extraites d'une table.
- Colonne avec accès séquentiel mais avec un where between, car un ordre est spécifié.

Pour quels types de requêtes un index Clustérisé serait conseillé ?

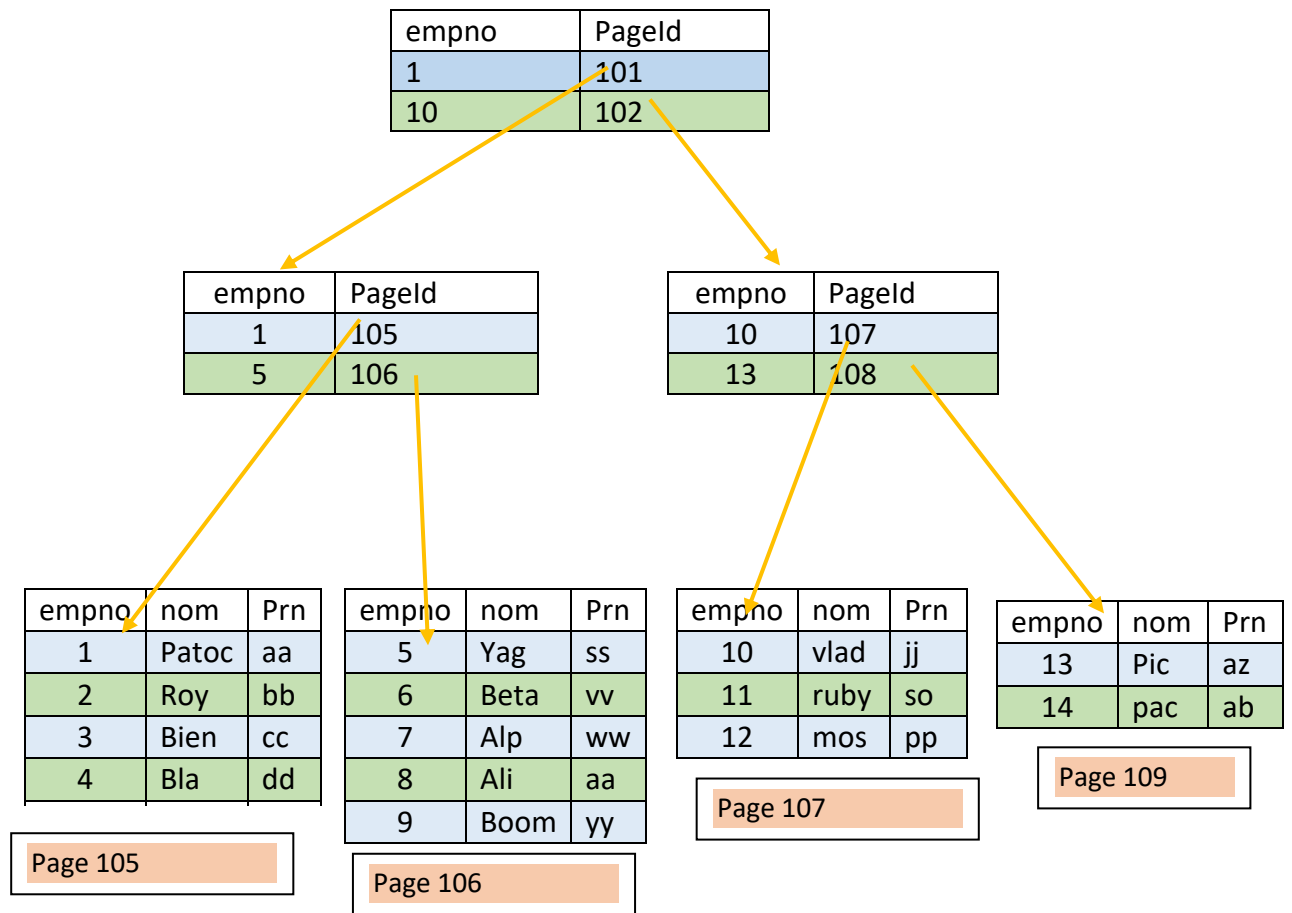
- Requêtes avec qui retournes une plage de valeurs : WHERE >, WHERE <, WHERE BETWEEN ..
- Retourne un résultat volumineux
- Pour les jointures
- Order by et group by

Éviter les index sur les colonnes :

- Très sujettes au changement : UPDATE
- Les clés étendues (clé composée et de types varchar)

Principe :

Dans les index Clustérisés le système est organisé sous forme d'arborescence binaire parfaitement équilibré. B-Arbre. Le parcours de l'arbre est suffisant pour obtenir toute l'information désirée. Voir exemple plus bas.



Les index non CLUSTERED INDEX :

Un index non-cluster contient les valeurs de clé d'index et les localisateurs de ligne qui pointent vers l'emplacement de stockage des données de table.

Vous pouvez créer plusieurs index non cluster sur une table ou une vue indexée.

Les index non-cluster doivent, en principe, améliorer les performances des requêtes fréquemment utilisées qui ne sont pas couvertes par l'index cluster.

La commande CREATE INDEX

Pour les index non cluster :

```
CREATE INDEX nomIndex ON nomTable(nomColonne);
```

Exemple :

```
CREATE INDEX typeIndex ON empClg(typeEmp);
```

Pour les index cluster :

```
CREATE CLUSTERED INDEX nomIndex ON nomTable(nomColonne);
```

Exemple

```
CREATE CLUSTERED INDEX INDXALIAS ON personnages(ALIAS);
```

En général :

```
CREATE [CLUSTERED] INDEX nom_de_index ON nom_table (nom_colonne)
```

```
CREATE INDEX nom_de_index ON nom_table (nom_colonne)
```

```
ALTER TABLE nom_table ADD CONSTRAINT nom_contrainte  
PRIMARY KEY (nom_colonne)
```

```
ALTER TABLE nom_table ADD CONSTRAINT nom_contrainte  
PRIMARY KEY NONCLUSTERED (nom_colonne)
```

```
ALTER TABLE nom_table ADD CONSTRAINT nom_contrainte  
UNIQUE (nom_colonne)
```

```
ALTER TABLE nom_table ADD CONSTRAINT nom_contrainte  
UNIQUE [CLUSTERED] (nom_colonne)
```

Suppression d'un index

```
DROP INDEX nom_index ON nom_table
```

Afficher les index définis sur une table

```
EXEC sys.sp_helpindex @objname = 'nom_table'
```

Outils de mesures des performances

Entrer la commande suivante pour activer les mesures des performances des requêtes

```
SET STATISTICS IO, TIME ON | OFF
```

On peut activer l'affichage du plan d'exécution des requêtes avec la commande Ctrl-M dans MS SQL Studio.

Règles d'optimisation de requêtes :

Lorsque vous écrivez vos requêtes, même si les SGBDs ont des optimiseurs, voici quelques règles à respecter pour optimiser vos requêtes. (qui ne sont pas nécessairement dans l'ordre).

- R1 : Éviter le SELECT * : écrire plutôt le nom des colonnes dont vous avez besoin pour la requête.
- R2 : Créez des indexes sur les colonnes que vous utilisez dans la clause WHERE. Pour plus de performances, **ces indexes doivent-être créés après l'insertion des données dans la table.**
- R3 : Lorsque c'est possible, utilisez le WHERE à la place du Having.
- R4 : Éviter les jointures dans le WHERE, utilisez plutôt le INNER JOIN.
- R5 : Lorsque c'est possible, utilisez une jointure à la place d'une sous-requête. Les jointures sont l'essentiel des SGBDRs alors ils sont optimisés pour l'écriture des jointures.

Chapitre 10, introduction à la sécurité de données

Introduction


Aucune méthode universelle n'existe pour créer une application cliente SQL Server sécurisée. Chaque application est unique au niveau de sa configuration, de son environnement de déploiement et de ses utilisateurs. Une application relativement sécurisée lors de son déploiement initial peut devenir moins sécurisée avec le temps. Il est impossible d'anticiper avec précision sur les menaces qui peuvent survenir dans le futur.

Menaces courantes :

Les développeurs doivent connaître les menaces de sécurité, les outils disponibles pour les contrer et la manière d'éviter les défaillances de sécurité qu'ils se créent eux-mêmes. La sécurité peut être envisagée comme une chaîne dans laquelle un maillon manquant compromet la solidité de l'ensemble. La liste suivante comprend quelques menaces de sécurité courantes évoquées plus en détail dans les rubriques de cette section.

Injection SQL

L'injection SQL est le processus qui permet à un utilisateur malveillant d'entrer des instructions Transact-SQL au lieu d'une entrée valide. Si l'entrée est transmise directement au serveur sans validation et si l'application exécute accidentellement le code injecté, l'attaque risque d'endommager ou de détruire des données.

Important : 

Vous pouvez déjouer les attaques d'injection SQL Server [à l'aide de procédures stockées et de commandes paramétrées](#), en évitant le code SQL dynamique et en limitant les autorisations de tous les utilisateurs :

Validez TOUTES les entrées.

Les Injection SQL peuvent se produire en modifiant une requête de façon à ce qu'elle soit toujours exécutée (retourne toujours vrai) en changeant la clause WHERE ou avec un opérateur UNION

Exemples:

SELECT * from utilisateurs where nom = @nom, en ADO.net

SELECT * from utilisateurs where nom = ? en PDO

En théorie cette requête ramène les informations (mot de passe) d'un utilisateur dont le nom est en paramètre, donc seules les personnes connaissant la valeur du paramètre nom pourront chercher les informations correspondantes

Imaginez maintenant que quelqu'un soit malintentionné remplace la requête par:

```
SELECT * from utilisateurs where nom = 'Patoche' OR 1=1;
```

Comme 1=1 est tout le temps vrai, alors la requête va renvoyer les informations de tous les utilisateurs.

Ou encore

```
SELECT Description FROM produits
WHERE Description like '%Chaises'
UNION ALL
SELECT username FROM dba_users
WHERE username like '%'
```

Si les deux requêtes précédentes étaient dans des procédures stockées, le problème ne se serait pas posé. La validation des entrées est ESSENTIELLE.

Élévation de privilège :

Les attaques d'élévation de privilège se produisent lorsqu'un utilisateur s'empare des privilèges d'un compte approuvé, un administrateur ou un propriétaire par exemple. Exécutez toujours le code sous des comptes d'utilisateurs disposant des privilèges minimums et attribuez uniquement les autorisations nécessaires

Attention : 

Évitez l'utilisation des comptes d'administrateur (comme Sa pour SQL Server, root pour MySQL et system pour Oracle) pour l'exécution du code.

Supprimer les comptes utilisateurs non utilisés

Supprimer les comptes utilisateurs par défaut

Donnez les privilèges selon les besoins.

Détection des attaques et surveillance intelligente

Une attaque de détection peut utiliser des messages d'erreur générés par une application pour rechercher des vulnérabilités dans la sécurité. Implémentez la gestion des erreurs dans tout le code de procédure pour éviter de retourner des informations d'erreurs SQL Server à l'utilisateur final.

Attention : 

Ne pas afficher de messages d'erreur explicites affichant la requête ou une partie de la requête SQL. Personnalisez vos messages erreur.

Mots de passe

De nombreuses attaques réussissent lorsqu'un intrus a su deviner ou se procurer le mot de passe d'un utilisateur privilégié. Les mots de passe représentent la première ligne de défense contre les intrus, la définition de mots de passe forts est donc un élément essentiel de la sécurité de votre système. Créez et appliquez des stratégies de mot de passe pour l'authentification en mode mixte.

Attention : 

Renforcer les mots de passe.

Utilisez la stratégie des mots de passe pour les comptes sa, root et system.

Supprimer les comptes sans mot de passe.

Rôles du serveur :

SQL Server fournit des rôles au niveau du serveur pour vous aider à gérer les autorisations sur les serveurs

SQL Server fournit neuf rôles serveur fixes. Les autorisations accordées aux rôles serveur fixes (à l'exception de **public**) ne peuvent pas être changées.

Les rôles du serveur sont attribués [aux connexions](#)

Rôles	Description
sysadmin	Les membres du rôle serveur fixe sysadmin peuvent effectuer n'importe quelle activité sur le serveur.
serveradmin	Les membres du rôle serveur fixe serveradmin peuvent modifier les options de configuration à l'échelle du serveur et arrêter le serveur.
securityadmin	Les membres du rôle serveur fixe securityadmin gèrent les connexions et leurs propriétés. Ils peuvent attribuer des autorisations GRANT, DENY et REVOKE au niveau du serveur. Ils peuvent également attribuer des autorisations GRANT, DENY et REVOKE au niveau de la base de données, s'ils ont accès à une base de données. En outre, ils peuvent réinitialiser les mots de passe pour les connexions SQL Server .
processadmin	Les membres du rôle serveur fixe processadmin peuvent mettre fin aux processus en cours d'exécution dans une instance de SQL Server.
setupadmin	Les membres du rôle serveur fixe setupadmin peuvent ajouter et supprimer des serveurs liés à l'aide d'instructions Transact-SQL. (L'appartenance au rôle sysadmin est nécessaire pour utiliser Management Studio.)
bulkadmin	Les membres du rôle serveur fixe bulkadmin peuvent exécuter l'instruction <code>BULK INSERT</code> .
diskadmin	Le rôle serveur fixe diskadmin permet de gérer les fichiers disque.
dbcreator	Les membres du rôle serveur fixe dbcreator peuvent créer, modifier, supprimer et restaurer n'importe quelle base de données.
public	Chaque connexion SQL Server appartient au rôle serveur public . Lorsqu'un principal de serveur ne s'est pas vu accorder ou refuser des autorisations spécifiques sur un objet sécurisable, l'utilisateur hérite des autorisations accordées à public sur cet objet. Vous ne devez affecter des autorisations publiques à un objet que lorsque vous souhaitez que ce dernier

	soit disponible pour tous les utilisateurs. Vous ne pouvez pas modifier l'appartenance au rôle public.
--	--

Rôles niveau bases de données :

Les rôles niveau base de données sont attribués à un utilisateur mappé sur la connexion

Roles	Description
db_owner	Les membres du rôle de base de données fixe db_owner peuvent effectuer toutes les activités de configuration et de maintenance sur la base de données et peuvent également supprimer la base de données dans SQL Server. (Dans SQL Database et SQL Data Warehouse, certaines activités de maintenance requièrent des autorisations de niveau serveur et ne peuvent pas être effectuées par db_owners .)
db_securityadmin	Les membres du rôle de base de données fixe db_securityadmin peuvent modifier l'appartenance au rôle pour les rôles personnalisés uniquement et gérer les autorisations. Les membres de ce rôle peuvent potentiellement élever leurs privilèges et leurs actions doivent être supervisées.
db_accessadmin	Les membres du rôle de base de données fixe db_accessadmin peuvent ajouter ou supprimer l'accès à la base de données des connexions Windows, des groupes Windows et des connexions SQL Server.
db_backupoperator	Les membres du rôle de base de données fixe db_backupoperator peuvent sauvegarder la base de données.
db_ddladmin	Les membres du rôle de base de données fixe db_ddladmin peuvent exécuter n'importe quelle commande DDL (Data Definition Language) dans une base de données.

db_datawriter	Les membres du rôle de base de données fixe db_datawriter peuvent ajouter, supprimer et modifier des données dans toutes les tables utilisateur.
db_datareader	Les membres du rôle de base de données fixe db_datareader peuvent lire toutes les données de toutes les tables utilisateur.
db_denydatawriter	Les membres du rôle de base de données fixe db_denydatawriter ne peuvent ajouter, modifier ou supprimer aucune donnée des tables utilisateur d'une base de données.
db_denydatareader	Les membres du rôle de base de données fixe db_denydatareader ne peuvent lire aucune donnée des tables utilisateur d'une base de données.

Privilèges sur les objets (tables, colonnes, lignes) :

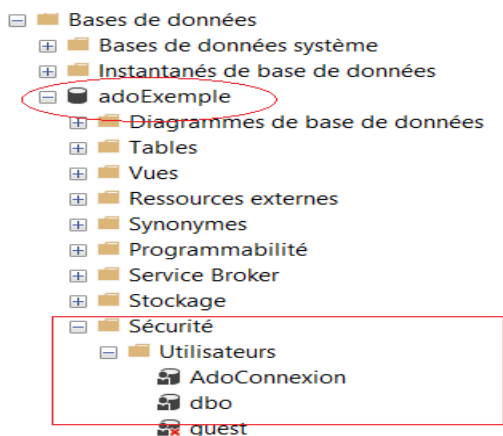
Par l'interface SQL Server Management Studio :

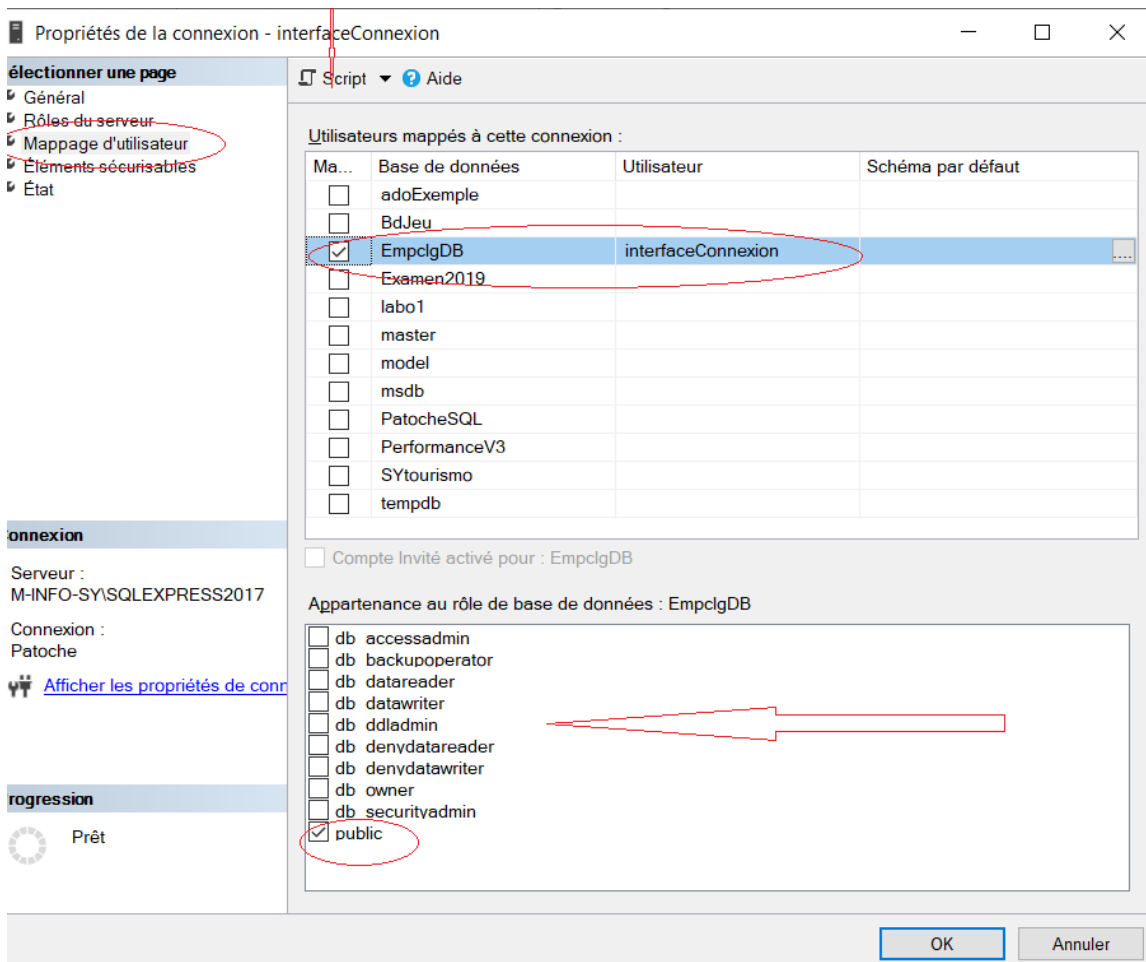
On suit les mêmes étapes pour créer une connexion, puis avant de cliquer sur OK, il faudra mapper un utilisateur à cette connexion

L'utilisateur mappé a le même nom que la connexion.

En général, les privilèges sont accordés aux utilisateurs (non aux connexions).

Un utilisateur utilise un login pour se connecter et il est rattaché à une base de données. Sinon, l'utilisateur est dit orphelin.





Dans cette figure on voit que :

- La connexion interfaceConnexion est mappée sur un utilisateur de même nom.
- La base de données de l'utilisateur est EmpclgDb
- Le role BD de l'utilisateur est public. Un role public ne donne aucun droit sur la BD. L'utilisateur peut faire un USE EmpclgDb et rien d'autre.

Si on clique pour générer le script on aura le script suivant :

```
USE [master]
GO
CREATE LOGIN [interfaceConnexion] WITH PASSWORD=N'123456',
DEFAULT_DATABASE=[EmpclgDB], CHECK_EXPIRATION=OFF,
CHECK_POLICY=OFF
GO
USE [EmpclgDB]
GO
CREATE USER [interfaceConnexion] FOR LOGIN [interfaceConnexion]
GO
```

Sélectionner une page

- Général
- Rôles du serveur
- Mappage d'utilisateur
- Éléments sécurisables
- État

Connexion

Serveur : M-INFO-SY\SQLSERVER2017
 Connexion : Patoche
 Afficher les propriétés de connexion

Progression

Prêt

Script Aide

Utilisateurs mappés à cette connexion :

Ma...	Base de données	Utilisateur	Schéma par défaut
<input type="checkbox"/>	adoExemple		
<input type="checkbox"/>	BdJeu		
<input checked="" type="checkbox"/>	EmpcigDB	PatocheUser	
<input type="checkbox"/>	Examen2019		
<input type="checkbox"/>	labo1		
<input type="checkbox"/>	master		
<input type="checkbox"/>	model		
<input type="checkbox"/>	msdb		
<input type="checkbox"/>	PatocheSQL		
<input type="checkbox"/>	PerformanceV3		
<input type="checkbox"/>	SYtourismo		
<input type="checkbox"/>	tempdb		

Compte Invité activé pour : EmpcigDB

Appartenance au rôle de base de données : EmpcigDB

<input type="checkbox"/>	db accessadmin
<input type="checkbox"/>	db backupoperator
<input checked="" type="checkbox"/>	db datareader
<input checked="" type="checkbox"/>	db datawriter
<input type="checkbox"/>	db ddladmin
<input type="checkbox"/>	db denydatareader
<input type="checkbox"/>	db denydatawriter
<input type="checkbox"/>	db owner
<input type="checkbox"/>	db securityadmin
<input checked="" type="checkbox"/>	public

Voici le script obtenu pour une connexion AdoConnexion par l'interface Management Studio

```
USE [master]
GO
CREATE LOGIN [AdoConnexion] WITH PASSWORD=N'Local$33',
DEFAULT_DATABASE=[adoExemple], CHECK_EXPIRATION=ON,
CHECK_POLICY=ON
GO
USE [adoExemple]
GO
CREATE USER [AdoConnexion] FOR LOGIN [AdoConnexion]
GO
USE [adoExemple]
GO
ALTER ROLE [db_datareader] ADD MEMBER [AdoConnexion]
GO
USE [adoExemple]
GO
ALTER ROLE [db_datawriter] ADD MEMBER [AdoConnexion]
GO
```

L'utilisateur AdoConnexion a le droit de faire select, update, insert et Delete sur toutes les tables de la base de données adoExemple. Mais, il ne peut pas créer des objets ou les altérer

Avec la connexion AdoConnexion, on peut faire :

```
select * from etudiants;
select * from programmes;
insert into programmes values (200, 'Mathématiques');
update programmes set nom_programme = 'Art moderne' where
code_prog = 412;
delete from etudiants where numad=1;
```

Avec la connexion AdoConnexion, on NE peut PAS faire :

```
create table cours (codeCours char(3) not null,
titre_cours varchar(30)not null,
constraint pk_cours primary key(codeCours));
```

Question : Quel est le rôle Base de données qu'on aurait dû attribuer à AdoConnexion pour que l'utilisateur puisse créer la table cours ?

Avec les commandes SQL

La commande CREATE LOGIN vous permet de créer une connexion (ce que nous avons fait avec l'interface graphique au début de la session.).

Évidemment, on pourra donner des ROLE sur le serveur à ce login, on y reviendra plus loin.

La commande CREATE USER ...nomUser . FOR LOGIN nomDuLogin permet de créer un utilisateur pour le login.

Si au moment de créer l'utilisateur, aucune base de données n'a été sélectionnée, on dira que l'utilisateur est orphelin.

Exemple : (**vous devez avoir le rôle sysadmin**)

```
create login logPatoche with password = 'alainPatoche$33';
create user PatocheUser for login logPatoche;
```

PatocheUser est un utilisateur orphelin. Aucun accès à aucune BD.

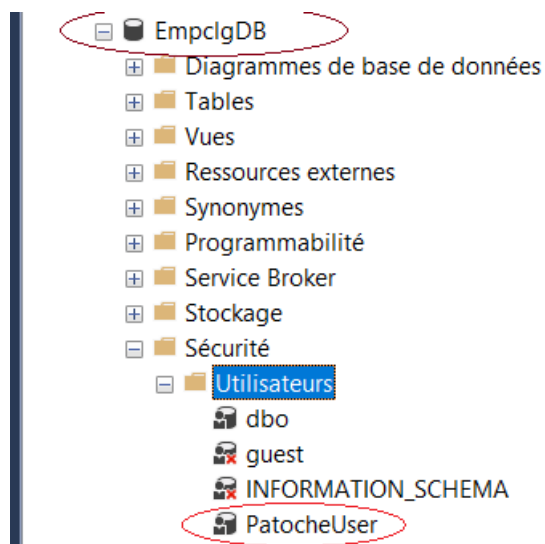
Pour créer un utilisateur non orphelin, donc rattachée à une base de données Empc1gDB par exemple, il faudra :

1. Avoir le role sysadmin
2. Faire un USE sur la BD Empc1gDB.

En faisant use Empc1gDB, puis CREATE USER, l'utilisateur crée est rattaché à la BD BdGestion. Mais il n'a aucun droit sur aucun objet de la BD.

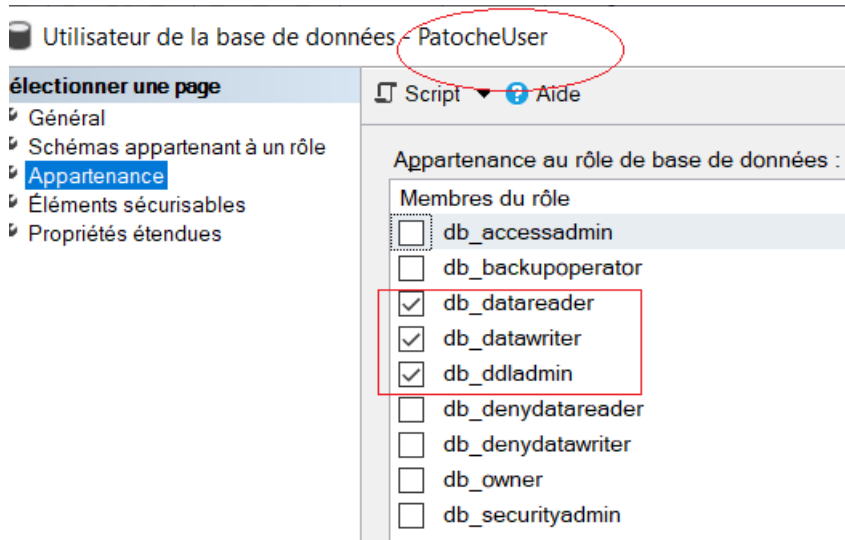
Exemple :

```
use Empc1gDB;  
create login logPatoche with password = 'alainPatoche$33';  
create user PatocheUser for login logPatoche;
```



Attribution des roles

```
ALTER ROLE [db_datareader] ADD MEMBER [PatocheUser];  
ALTER ROLE [db_datawriter] ADD MEMBER [PatocheUser];  
ALTER ROLE [db_ddladmin] ADD MEMBER [PatocheUser];
```



Avec ces role l'utilisateur PatocheUser peut faire, entre autres :

```
select * from livres;

update livres set titre = 'Comptabilité' where coteLivre = 'IF004';
insert into livres values('IM03', 'Ce livre', 'Italien', 2017, 890);

create table TabledePatoche
(
id_Personne int identity(1,1) ,
nom varchar(20) not null,
constraint pk_personne primary key (id_Personne)
);

insert into TabledePatoche values('Patoche');
```

Attention : 

Lorsque vous donnez des droits sur la base de données, ces droits s'appliquent à toutes **les table** de votre base de données.

Attention : 

Les droits de PatocheUser se limitent aux TABLES est non aux procédures stockées.

Ce qui veut dire que PatocheUser peut faire ceci :

```
update empClg set prenom = 'le roy' where nom = 'aa';
```

Et Ne peut PAS faire (Car il n'a pas le droit).

```
execute majPrenom  
@nom = 'aa',  
@prenom = 'Le roy';
```

Si on veut donner plus de privilèges à PatocheUser, on peut procéder avec la commande **GRANT**

```
grant execute on majPrenom to PatocheUser;
```

`grant execute to PatocheUser;` permet à PatocheUser d'exécuter n'importe quelle procédure.

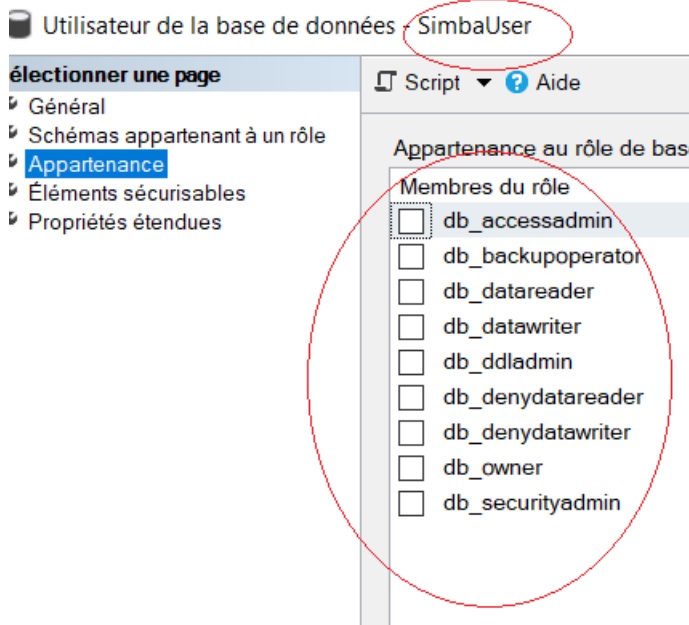
Les commandes GRANT, REVOKE et DENY

La command GRANT, syntaxe simplifiée

La commande GRANT permet d'attribuer des privilèges ou des permissions à un utilisateur sur un objet. Au lieu de donner des droits sur toutes les tables de la base de données par attribution de ROLE, on utilise la commande GRANT pour cibler les objets de la base de données qui seront affectés et restreindre les privilèges sur les objets ciblés pour les utilisateurs.

Exemple, nous souhaitons donner le droit SLECT sur notre table Questions à un autre utilisateur, qui est UserSimba. UserSimba est un utilisateur lié à une connexion et une base de données avec aucun role sur le serveur ni sur la base de données (il a le role Public) car il a été créé comme suit :

```
use BdJeu;  
create login logSimba with password = 'Simbaleroy$22';  
create user SimbaUser for login logSimba;
```



On peut cependant permettre certaines actions sur les table pour le user UserSimab

Syntaxe simplifiée de la commande GRANT

```
GRANT { ALL [ PRIVILEGES ] }
      | permission [ ( column [ ,...n ] ) ] [ ,...n ]
      [ ON [ class :: ] securable ] TO principal [ ,...n ]
      [ WITH GRANT OPTION ] [ AS principal ]
```

Le ALL est à déconseiller. Il vaut mieux attribuer les autorisations ou les privilèges au besoin

- Si l'élément sécurisable est une fonction scalaire, ALL représente EXECUTE et REFERENCES.
- Si l'élément sécurisable est une fonction table, ALL représente DELETE, INSERT, REFERENCES, SELECT et UPDATE.
- Si l'élément sécurisable est une procédure stockée, ALL représente EXECUTE.
- Si l'élément sécurisable est une table, ALL représente DELETE, INSERT, REFERENCES, SELECT et UPDATE.
- Si l'élément sécurisable est une vue, ALL représente DELETE, INSERT, REFERENCES, SELECT et UPDATE.

Exemples :

```
Grant select, insert on categories to SimbaUser;  
grant select, insert, update(enonce) on Questions to SimbaUser;  
grant select on personnes to SimbaUser with grant option;
```

WITH GRANT OPTION, signifie que l'utilisateur qui a reçu le privilège peut donner le même privilège à un autre utilisateur.

Les rôles créés par les utilisateurs. (pas ceux prédéfinis).

On peut créer des RÔLES. Un rôle va regrouper plusieurs privilèges. L'avantage d'avoir des rôles, c'est de donner le même rôle à plusieurs utilisateurs. De plus..au lieu d'ajouter un GRANT pour un user, il suffit de l'appliquer au RÔLE.

Situation : vous êtes une équipe de 10, donc **10 users** à travailler sur le même projet. Vous utilisez donc des tables de ce projet. Ces tables ne vous appartiennent pas mais vous y avez accès avec des autorisations

En tant que propriétaire de la BD BdJeu :

```
use BdJeu;  
create role roleprojet;  
Grant select, insert on categories to roleprojet;  
grant select, insert, update(enonce) on Questions to roleprojet;
```

Je viens de créer un rôle roleprojet avec un certain nombre de droits.

Tout ce que nous avons à faire c'est d'affecter le rôle aux usagers qu'on veut.

```
EXEC sp_addrolemember roleprojet, RubyUser;  
EXEC sp_addrolemember roleprojet, RemiUser;
```

Maintenant... imaginez que vous avez oublié de donner le privilège SELECT sur la table joueurs pour les 10 users de l'équipe de projet. Comment allez-vous faire ? et surtout comment ne pas oublier aucun utilisateur ?

Il suffit de faire un GRANT pour votre role. De cette façon, tous les users qui on eu le role se verront GRANTÉ, le privilège.

```
grant select on joueurs to roleprojet;
```

La commande REVOKE.

La commande REVOKE permet de retirer des droits. (Des privilèges) . En principe les privilèges ont été attribuer par la commande GRANT

Syntaxe :

```
REVOKE [ GRANT OPTION FOR ] <permission> [ ,...n ] ON
  [ OBJECT :: ] [ schema_name ]. object_name [ ( column [ ,...n
] ) ]
  { FROM | TO } <database_principal> [ ,...n ]
  [ CASCADE ]
  [ AS <database_principal> ]
```

Exemple :

```
revoke insert on Categories from SimbaUser;
```

La commande DENY

Il arrive qu'un utilisateur, ait hérité des droits car il est membre d'un role. Une façon de ne pas autoriser (d'interdire) l'utilisateur en question à ne pas faire certaines opérations c'est avec la commande DENY

Syntaxe :

```
DENY { ALL [ PRIVILEGES ] }
  | <permission> [ ( column [ ,...n ] ) ] [ ,...n ]
  [ ON [ <class> :: ] securable ]
  TO principal [ ,...n ]
  [ CASCADE ] [ AS principal ]
[;]
```

Les vues pour la sécurité des données : contrôle sur les lignes

Nous avons abordé les vues comme étant des objets de la base de données permettant la simplification de requêtes. Dans ce qui suit, nous allons voir comment les vues peuvent contribuer à la sécurité des données.

Les vues permettent de protéger l'accès aux tables en fonction de chacun des utilisateurs. On utilise une vue sur une table et on interdit l'accès aux tables. C'est donc un moyen efficace de protéger les données.

```
CREATE [ OR ALTER ] VIEW [ schema_name . ] view_name
AS select_statement
[ WITH CHECK OPTION ]
[ ; ]
```

L'option WITH CHECK OPTION permet d'assurer que les modifications apportées à la table (dont la vue est issue) via la vue respectent la CLAUSE WHERE. Lorsqu'une ligne est modifiée via la vue, alors les données devraient rester cohérentes.

Exemple :

```
create view VSport as select * from questions where
code_categorie =3 with check option;
grant select, update, insert on Vsport to user1;
```

L'instruction suivante exécutée par le user1 va marcher car le code catégorie est 3:

```
insert into VSport values('une question',1,'facile',3);
```

L'instruction suivante exécutée par le user1 NE va PAS marcher car le code catégorie est 2. Ne respecte pas la clause WHERE:

```
insert into VSport values('une autre question',1,'facile',2);
```

De plus, le USER1, ne voit pas TOUT le contenu de la table Questions

Conclusion

Voici ce qu'il faudra retenir pour l'instant pour la sécurité des données :

1. Utilisez des procédures stockées.
2. Renforcer les mots de passes des comptes utilisateurs
3. Évitez d'utiliser les comptes des supers usagers (root, sa, system..) pour les opérations courantes.
4. Restreindre au minimum les autorisations, les privilèges pour les comptes utilisateurs
5. Supprimer les comptes utilisateurs par défaut. (anonymous, Guest, scott...)
6. Éviter les comptes sans mot de passe
7. Valider toutes les entrées. Éviter les chaînes null, drop, or, where ...
8. Vérifier le format des données saisie.
9. N'afficher jamais les messages erreurs renvoyés par le SGBD. Personnalisez vos messages erreurs.

Le chiffrement des données

Définition :

Le chiffrement est un procédé de la cryptographie qui consiste à rendre les données illisibles ou impossibles à lire sauf si vous avez une clé de déchiffrement.

Deux techniques peuvent être utilisées pour chiffrer les données

Hachage « hashing » (chiffrement unidirectionnel)

- La technique de hachage des données a la particularité d'être **irréversible**; il n'est pas possible de retrouver les données originales après avoir été crypté avec une fonction de hachage.
- Il s'agit d'une fonction mathématique qui prend en entrée des données (chaîne de caractères de longueur variable) et qui génère en sortie une chaîne de caractères de longueur fixe appelée « hash »;
- La sortie (hash) est toujours la même pour des entrées identiques;
- Cette technique de chiffrement est couramment utilisée pour conserver des mots de passe ou vérifier l'intégrité d'un document;
- Algorithmes de hachage les plus utilisés;
 - SHA2_256, SHA2_512
- Dans **MS SQL Server** le chiffrement par hachage est fait avec la fonction **HASHBYTES**;
 - La fonction prend deux paramètres
 - L'algorithme à utiliser et les données à chiffrer;
 - L'algorithme ne chiffre pas des chaînes plus longues que 8000 caractères;

Chiffrement des données (chiffrement bidirectionnel)

Chiffrement symétrique

- Méthode de chiffrement rapide qui utilise une seule clé; la même clé est utilisée pour crypter et décrypter les données;
- Algorithmes de chiffrement symétrique les plus utilisés;
 - AES_128, AES_192 et AES_256;
 - Supporté par MS SQL Server;
 - Considéré le plus sécuritaire aujourd'hui;
 - Choisi par le gouvernement américain pour remplacer l'algorithme de chiffrement DES;

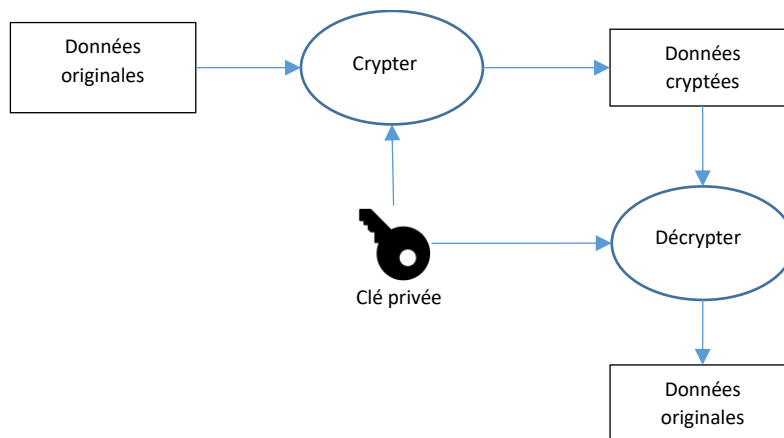


Figure 2: Chiffrement symétrique

Chiffrement asymétrique (https://en.wikipedia.org/wiki/Alice_and_Bob)

- Méthode de chiffrement relativement lente qui utilise deux clés, une clé publique et une clé privée;
- Comme leur nom l'indique, la clé publique peut être distribuée librement à quiconque souhaite communiquer de manière confidentielle avec le détenteur de la clé privée;
- Les données cryptées avec la clé publique peuvent être décryptées uniquement avec la clé privée;
- L'inverse est aussi vrai, les données cryptées avec la clé privée peuvent être décryptées par tous ceux qui possèdent la clé publique;
- Algorithmes de chiffrement asymétrique les plus utilisés;
 - RSA (Rivest, Shamir et Adleman), DSA (Digital Signature Algorithm);

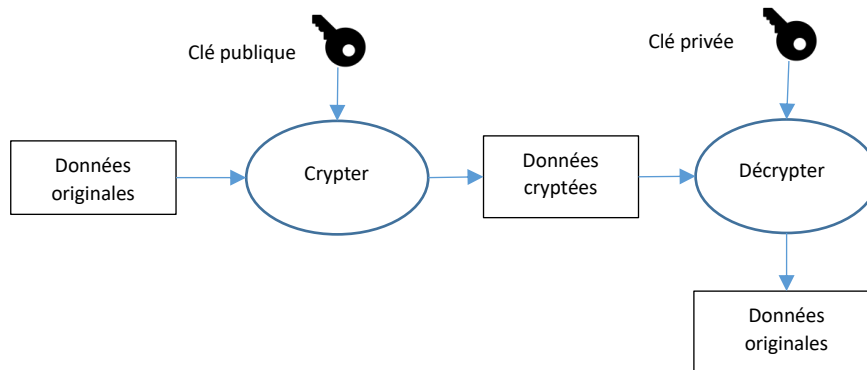


Figure 3: Chiffrement asymétrique

Chiffrement des procédures et fonctions de la base de données

Il est possible de chiffrer les procédures stockées ainsi que les fonctions. Après avoir crypté une procédure, il n'est plus possible d'en voir le texte. Une situation qui pourrait demander de crypter les procédures et fonctions est si la BD est livrée chez un client qui n'a pas payé pour le code source.

Il est fortement recommandé de garder une copie de la BD dans laquelle les procédures ne sont pas cryptées puisque la procédure pour les récupérer n'est si simple.

Il semblerait que cette méthode de chiffrement des procédures et fonctions ait été compromise et que plusieurs logiciels commerciaux sont disponibles pour récupérer le code de ces procédures;

```

create procedure ValiderMotDePasse(@motDePasse varchar(60))
with encryption
as
begin
end
  
```

Figure 4: Procédure chiffrée

Chiffrer les données contenues dans une table

Le chiffrement des données destinées à être conservées dans la base de données peut se faire soit dans le logiciel client (la page Web ou l'application Form) ou dans le SGBD.

Dans le cas d'une application web, le chiffrement des données se fait typiquement dans le serveur d'application web. Dans le cas d'un programme Form, chaque programme est responsable du chiffrement des données.

Chiffrement des données dans le SGBD MS SQL Server

MS SQL Server offre des fonctions permettant de crypter les données dans des procédures stockées.

Notez que toutes les fonctions de chiffrements documentées dans **MS SQL Server** ne peuvent pas être utilisées avec le **SGBD Microsoft Azure**.

Les logiciels qui utilisent une base de données Azure n'ont d'autres choix que de gérer le chiffrement des données dans le logiciel client ou dans le cas d'application web dans le serveur d'application web.

Chiffrement symétrique

ENCRYPTBYKEY, DECRYPTBYKEY

Cette fonction utilise une clé privée pour chiffrer les données. Cette clé doit être préalablement créée avec la commande 'CREATE SYMMETRIC KEY'. C'est au moment de créer la clé symétrique que l'on peut spécifier l'algorithme de chiffrement.

```
create symmetric key cle_sym with algorithm = AES_256
    encryption by password = 'Mon mot de passe robuste'
go

open symmetric key cle_sym
    decryption by password = 'Mon mot de passe robuste'
go

declare @info_a_chiffrer varchar(500)
declare @info_chiffree varbinary(8000)

set @info_a_chiffrer = 'Les carottes sont cuites'

print @info_a_chiffrer
print datalength( @info_a_chiffrer )

set @info_chiffree = EncryptByKey( key_guid('cle_sym'),
@info_a_chiffrer )

print @info_chiffree
print datalength( @info_chiffree )

select convert(varchar, DecryptByKey(@info_chiffree))
```

Figure 5: Exemple de chiffrement symétrique avec la commande **EncryptByKey**

ENCRYPTBYPASSPHRASE, DECRYPTBYPASSPHRASE

Ces fonctions utilisent une clé privée pour chiffrer les données. La clé est fournie sous la forme d'une chaîne de caractères (un mot de passe par exemple). La fonction utilise à l'interne la commande 'CREATE SYMMETRIC KEY' pour créer une clé symétrique.

```

create table joueurs
(
  no_jou      int identity(1, 1)
  constraint joueurs_pk primary key,
  alias_j     varchar(20) unique,
  prenom_j    varchar(20),
  nom_j       varchar(20),
  carte_credit varbinary(8000) null,
  mot_passe   varbinary(8000) null
)

-- Le numéro de carte de crédit '6544897' est
-- chiffré avec la clé 'Pa$$w0rd'
update joueurs
set carte_credit = ENCRYPTBYPASSPHRASE('Pa$$w0rd', '6544897', 0)
where alias_j = 'Wi'

-- Le numéro de carte de crédit est déchiffré et convertit en varchar
select convert(varchar, DECRYPTBYPASSPHRASE('Pa$$w0rd', carte_credit,
0))
from joueurs

```

Figure 6: exemple de chiffrement symétrique avec **ENCRYPTBYPASSPHRASE**.

Fonction de hachage

HASHBYTES

Cette fonction prend en paramètre la chaîne de caractères à chiffrer et l'algorithme de chiffrement. La fonction retourne une chaîne chiffrée.

```

declare @crypt_pass varbinary(8000);
select @crypt_pass = HASHBYTES('SHA2_512', 'pass123');

select HASHBYTES('SHA2_512', 'pass123');

```

Figure 7: Exemple de hachage de données

Chiffrement des données dans le logiciel client ou le serveur d'application web

Traiter le chiffrement des données du côté client a l'avantage que l'information confidentielle est transmise chiffrée au SGBD. Cette approche ne nécessite pas d'avoir un canal de communication cryptée pour communiquer avec le SGBD.

Le "Framework .Net" offre tous les services de chiffrements dans le domaine «System.IO.Cryptography »

```
static string FonctionDeHachage(string infoAHacher)
{
    UnicodeEncoding UnicodeString = new UnicodeEncoding();

    Byte[] MotDePasseAChiffrer = UnicodeString.GetBytes(infoAHacher);

    MD5CryptoServiceProvider MD5 = new MD5CryptoServiceProvider();

    byte[] infoHachee = MD5.ComputeHash(MotDePasseAChiffrer);

    return Convert.ToBase64String(infoHachee);
}
```

Autre exemple chiffrement par clé symétrique sans certificat

```
create database ExempleEncryption;
USE ExempleEncryption;

drop table Clients3
create table Clients3
(
id_client int identity(1,1) constraint pkClient3 primary key,
nom varchar(30) not null,
prenom varchar(30) not null,
carteCredit varchar(20)
);

insert into Clients3 values ('Roy', 'Simon', '9874-1234-5678-1111');
insert into Clients3 values ('Lechat', 'Ryby', '1234-9874-2222-4569');
insert into Clients3 values ('Patouche', 'Mosus', '2222-3333-4444-5555');

-----On ajoute une colonne qui va contenir les données chiffrées
alter table Clients3 add CarteEncryptee varbinary(max) null;
```

-----Créer une clé symétrique encryptée par mot de passe.

```
USE ExempleEncryption;
CREATE SYMMETRIC KEY SymmetricKey2
WITH ALGORITHM = AES_128
ENCRYPTION BY password = 'CemotdePasse123';
GO

USE ExempleEncryption;
GO
--ouverture de la clé pour encryption
OPEN SYMMETRIC KEY SymmetricKey2
Decryption BY password = 'CemotdePasse123';
GO
On met à jour la colonne à chiffrer

UPDATE Clients3 SET CarteEncryptee = EncryptByKey
(Key_GUID('SymmetricKey2'), carteCredit)

GO
--fermer la clé de chiffrement
CLOSE SYMMETRIC KEY SymmetricKey2;
GO
```

--on affiche pour voir que les données ont été cryptée.
A ce stade... on aurait pu supprimer la colonne CarteCredit pour plus de sécurité.

```
select * from Clients3;
```

	id_client	nom	prenom	carteCredit	CarteEncryptee
1	1	Roy	Simon	9874-1234-5678-1111	0x00430AC8B2F56743B07CE8BD8181BC560200000B977BA8...
2	2	Lechat	Ryby	1234-9874-2222-4569	0x00430AC8B2F56743B07CE8BD8181BC560200000EC9B92...
3	3	Patouche	Mosus	2222-3333-4444-5555	0x00430AC8B2F56743B07CE8BD8181BC560200000375811A...

-on utilise à nouveau la clé pour décrypter

```

USE ExempleEncryption;
GO
OPEN SYMMETRIC KEY SymmetricKey2
Decryption BY password = 'CemotdePasse123';
GO

-- on affiche les truc déchiffré
SELECT nom, prenom, CarteEncrypatee AS 'Carte cryptée',
CONVERT(varchar, DecryptByKey(CarteEncrypatee)) AS 'Carte
déchiffré' FROM Clients3;

```

Résultats		Messages		
	nom	prenom	Carte cryptée	Carte déchiffré
1	Roy	Simon	0x00430AC8B2F56743B07CE8BD8181BC5602000000B977BA8...	9874-1234-5678-1111
2	Lechat	Ryby	0x00430AC8B2F56743B07CE8BD8181BC5602000000EC9B92...	1234-9874-2222-4569
3	Patouche	Mosus	0x00430AC8B2F56743B07CE8BD8181BC5602000000375811A...	2222-3333-4444-5555

Pour effectuer une insertion, on fait comme suit :

```

USE ExempleEncryption;
GO
--ouverture de la clé pour encryption
OPEN SYMMETRIC KEY SymmetricKey2
Decryption BY password = 'CemotdePasse123';
GO
insert into Clients3 values('Blabla', 'un nom', '9999-9999-9999-
9999', EncryptByKey (Key_GUID('SymmetricKey2'), '9999-9999-9999-9999' ));
GO
--fermer la clé
CLOSE SYMMETRIC KEY SymmetricKey2;
GO

```

Autre exemple chiffrement par ENCRYPTBYPASSPHRASE

```

create table Clients2
(
id_client int identity(1,1) constraint pkClient2 primary key,
nom varchar(30) not null,
prenom varchar(30) not null,
carteCredit varchar(20)
);

```

```

insert into Clients2 values ('Roy', 'Simon', '9874-1234-5678-1111');
insert into Clients2 values ('Lechat', 'Ryby', '1234-9874-2222-4569');
insert into Clients2 values ('Patouche', 'Mosus', '2222-3333-4444-5555');

alter table Clients2 add CarteEncryptee varbinary(max) null;

```

Chiffrement de la colonne ajoutée

```

update Clients2 set CarteEncryptee=
ENCRYPTBYPASSPHRASE('passord123456', '9874-1234-5678-1111') where
id_client=1;
update Clients2 set CarteEncryptee=
ENCRYPTBYPASSPHRASE('password123456', '1234-9874-2222-4569') where
id_client=2;
update Clients2 set CarteEncryptee=
ENCRYPTBYPASSPHRASE('pasord123456', '9874-1234-5678-1111') where
id_client=3;

```

Insertion en utilisant le chiffrement

```

insert into clients2 values('Test', 'Test', '0000-0000-0000-0000',
ENCRYPTBYPASSPHRASE('local$33', '0000-0000-0000-0000'));

```

Decryption

```

USE ExempleEncryption;

SELECT nom, prenom, CarteEncryptee AS 'carte encryptee',
CONVERT(varchar, DECRYPTBYPASSPHRASE('passord123456', CarteEncryptee))
AS 'carte decryptée' FROM Clients2 where id_client =1;

```

Sources

<https://docs.microsoft.com/en-us/sql/t-sql/functions/fetch-status-transact-sql?view=sql-server-2017>

<https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-2017>

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-2017>

<https://docs.microsoft.com/en-us/sql/?view=sql-server-2017>

<https://docs.microsoft.com/fr-fr/sql/2014-toc/sql-server-transaction-locking-and-row-versioning-guide?view=sql-server-2014>

<https://docs.microsoft.com/fr-fr/sql/2014-toc/sql-server-index-design-guide?view=sql-server-2014#Clustered>

<https://docs.microsoft.com/fr-fr/sql/relational-databases/security/authentication-access/server-level-roles?view=sql-server-ver15>

<https://docs.microsoft.com/fr-fr/sql/relational-databases/security/security-center-for-sql-server-database-engine-and-azure-sql-database?view=sql-server-ver15>

<https://docs.microsoft.com/fr-fr/dotnet/framework/data/adonet/sql/application-security-scenarios-in-sql-server>

https://www.ibm.com/support/knowledgecenter/fr/SSKLLW_9.5.0/com.ibm.bigfix.inventory.doc/Inventory/admin/t_sql_backup.html