Collège Lionel-Groulx

Département d'informatique

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

Automne 2023

Programmation de bases de données

Transact-SQL (SQL Server)

Saliha Yacoub

COLLEGE LIONEL-GROULX

Table des matières

Historique des versions

Numéro de version	Tâches/modifications	Auteur	Date
1.0	Chapitres 1,2,3,4,6	Saliha Yacoub	Août 2019
1.1	Chapitre 7,8	Saliha Yacoub	Octobre 2019
1.2	Chapitre 5	Saliha Yacoub Marc Beaulne	Octobre 2019
1.3	Chapitre 10	Saliha Yacoub	Novembre 2019
1.4	Chapitre 11	Saliha Yacoub	Novembre 2019
	Chapitre 11, le chiffrement	Marc Beaulne, Saliha Yacoub	Novembre 2019
1.5	Création du chapitre 9 : Les curseurs	Saliha Yacoub	Août 2020
2.0	Réorganisation des chapitres, ajouts de contenus, d'exemples	Saliha Yacoub	Août 2020
2.1	Ajout d'exemples :	Saliha Yacoub	Septembre 2021

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 2022. 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'incrémentation 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 et 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 nomdelaBD;
```

  Exemple :

```
CREATE DATABASE empclg;
```

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

```
USE nomdelaBD;
```

Exemple

```
USE empclg;
```

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

- 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, 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.

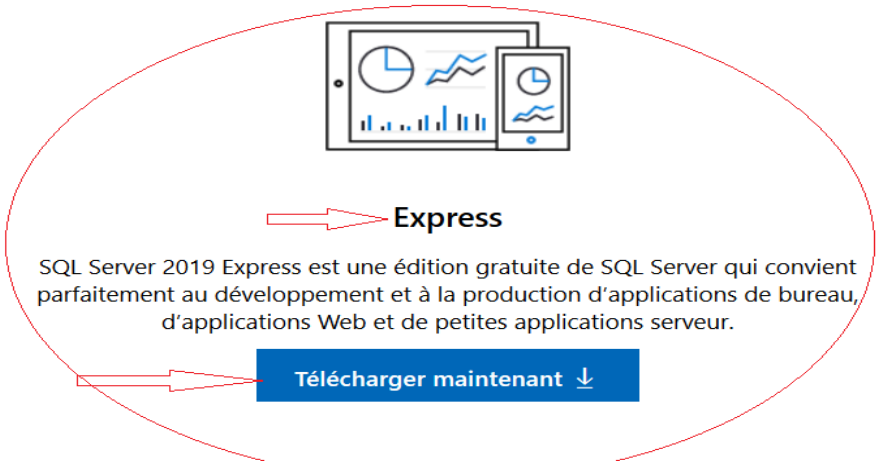
Nous avons besoin d'installer :

1- Le serveur de bases de données :
Vous devez aller sur le site suivant pour télécharger et installer SQL Server Express 2019.

https://www.microsoft.com/fr-ca/sql-server/sql-server-downloads

une édition spécialisée gratuite

net aux
n



Vous devez choisir installation « De base », et tout se déroule automatiquement.

Attention ! vous devez vérifier les paramètres de langue de votre ordinateur. **(La langue d'affichage de Windows doit correspondre à la langue du format régional)**

2- L'outil de gestion de bases de données

Une fois que le serveur est installé, vous devez installer SSMS (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://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

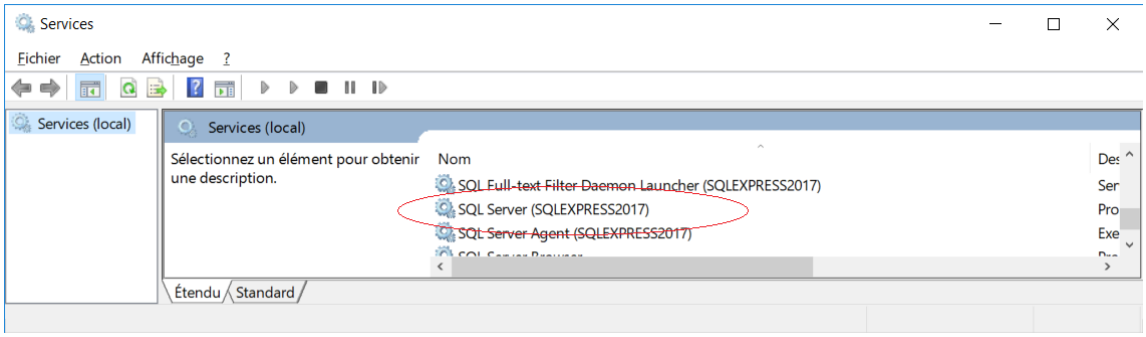
Vous pouvez choisir la version en **français** dans la section **Available Language**

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 :

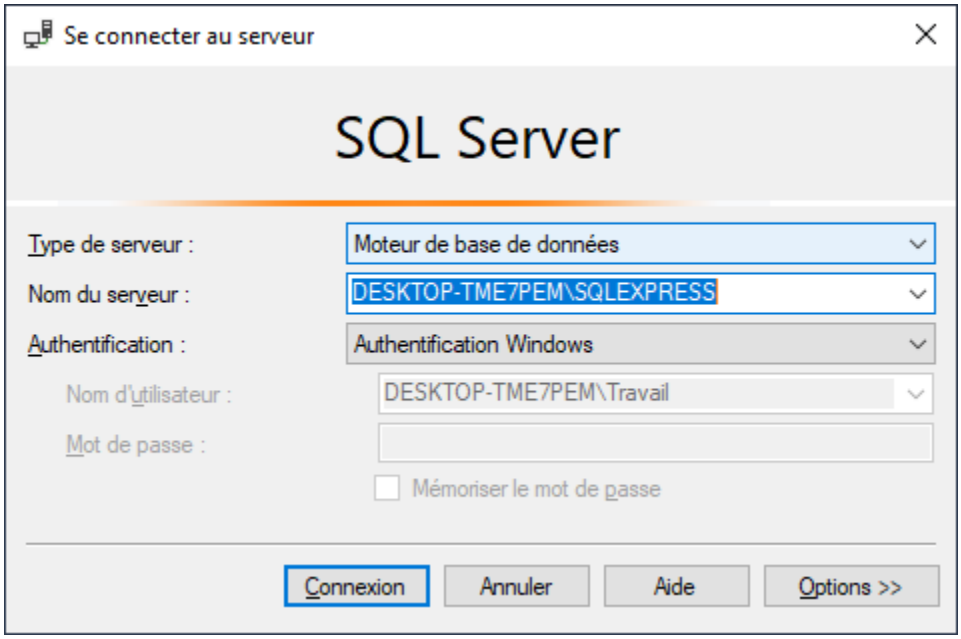
Authentification Windows : Si vous choisissez ce mode d'authentification, cela veut dire que le serveur de base de données, lorsque vous essayez de vous connecter, ne vous demandera pas de mot de passe. Utiliser ce mode d'authentification si vous n'avez pas de compte sur le serveur de base de données.

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'usager, d'un mot de passe et d'une base de données (si vous n'avez pas les droits pour vous en créer une). 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.

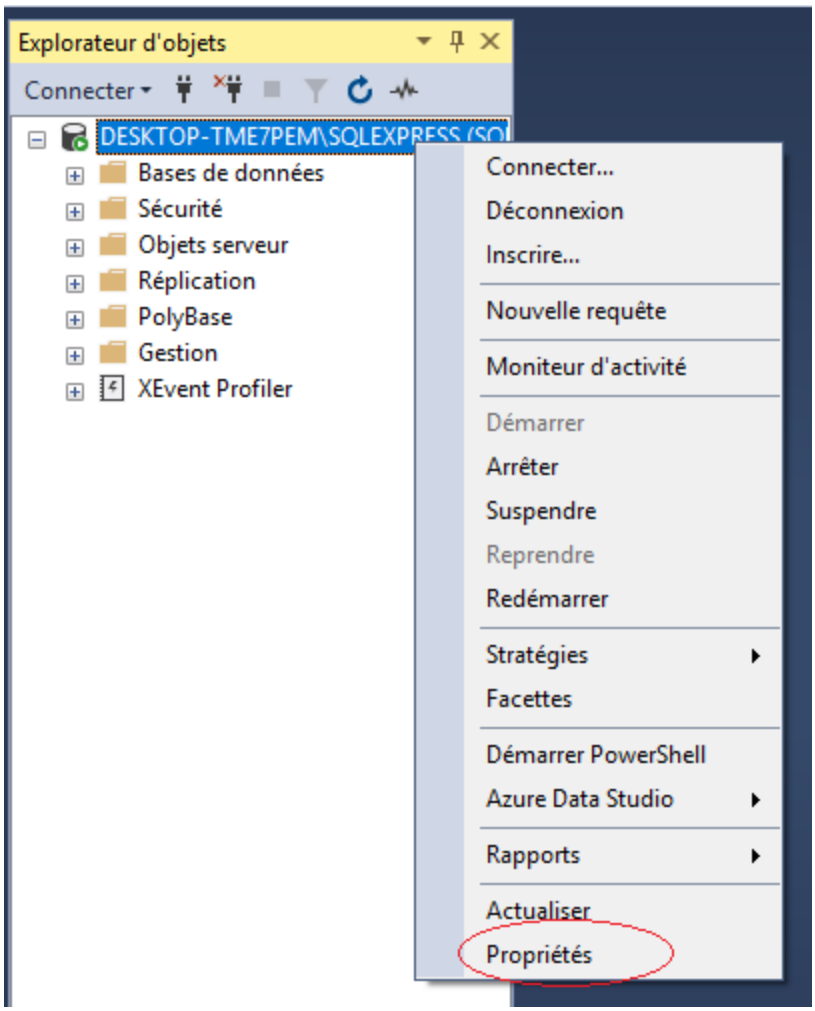


Attention : 

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

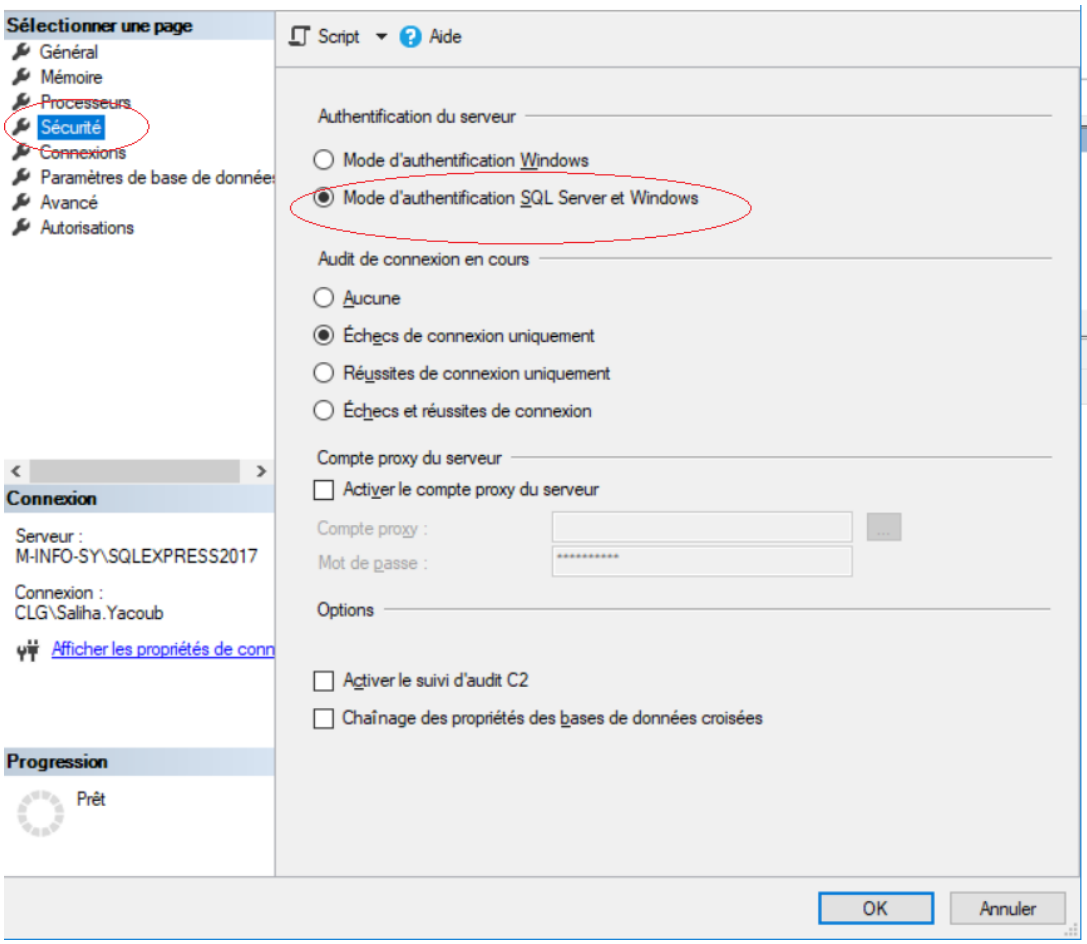
Sinon, dérouler le nom du serveur, faire parcourir (ou <Browse for more …> et trouver votre serveur et son instance.

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



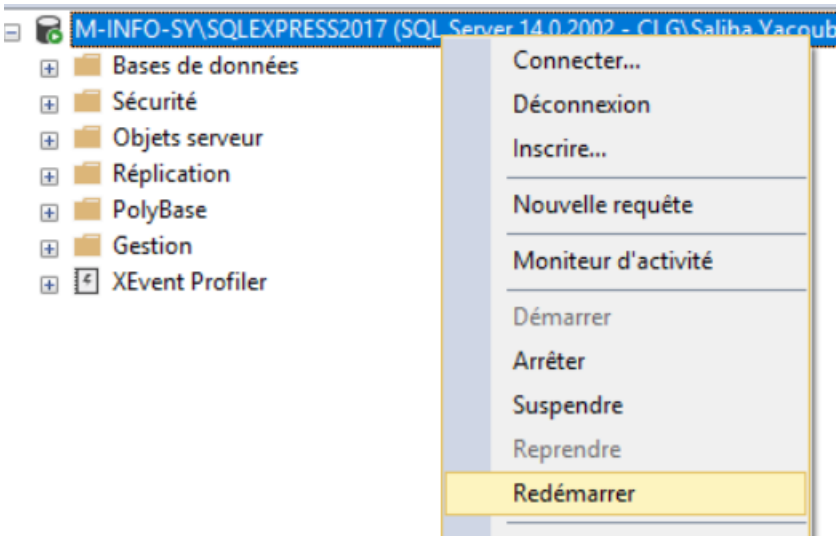
Bouton droit sur votre serveur, puis propriété Sécurité

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

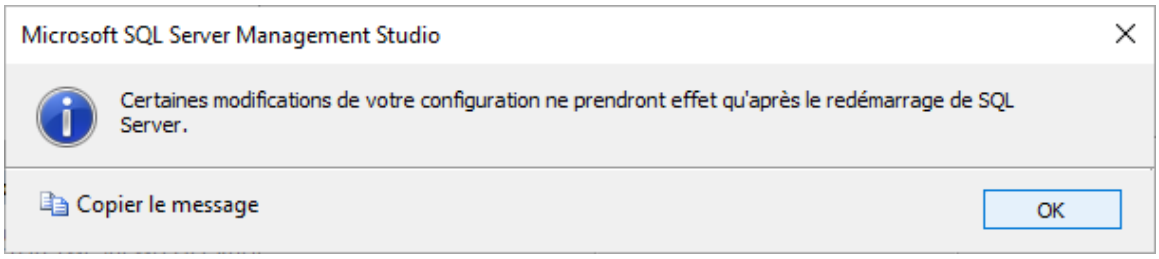


Attention :

Vous devez redémarrer le serveur

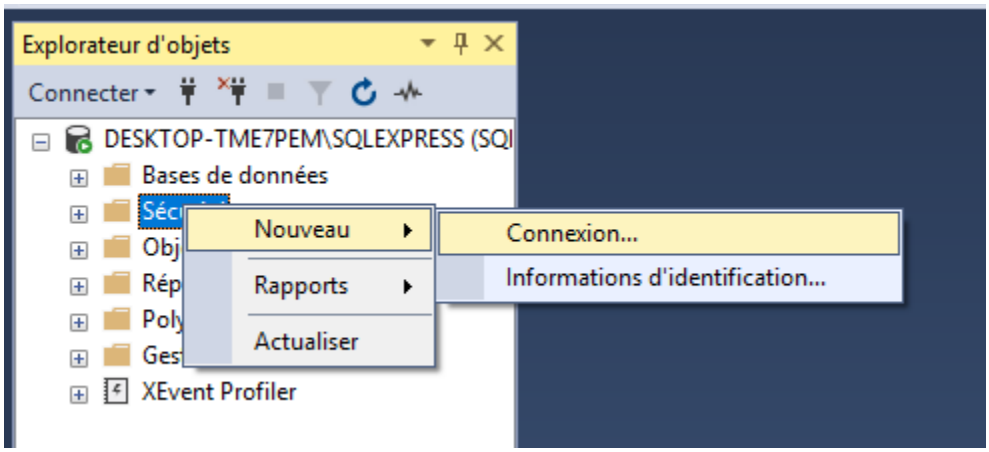


Il est probable que le serveur vous fasse une mise en garde quant au changement du mode d'authentification et qu'il faut redémarrer le serveur. Faîtes juste OK.

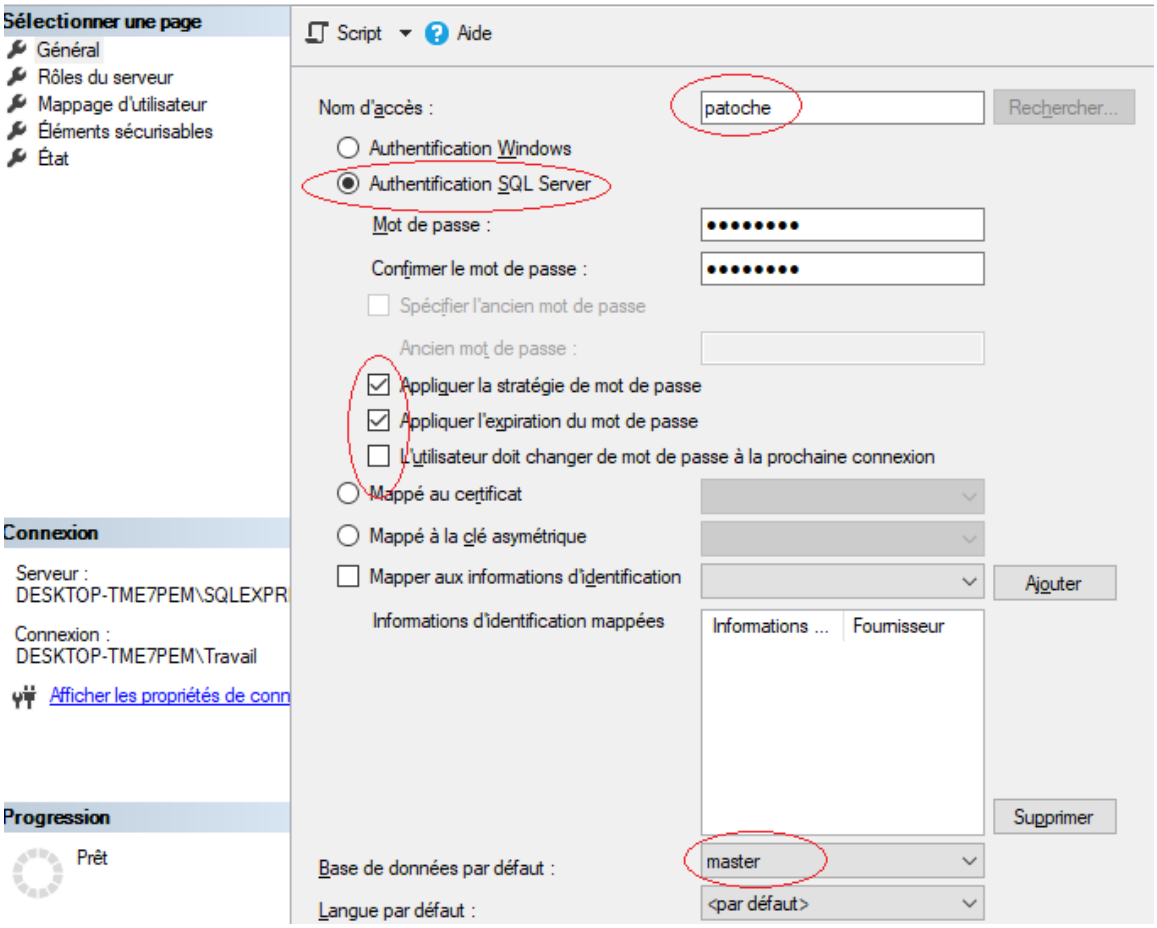


Étape 2 : Créer une nouvelle connexion

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

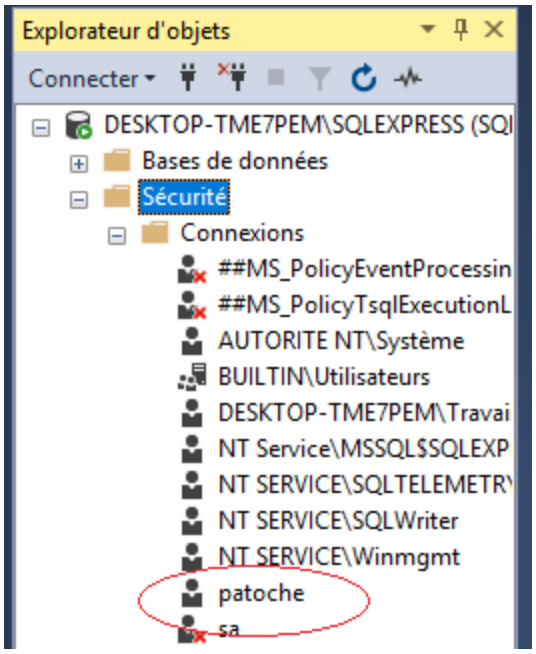


- Donner un nom significatif sans caractères spéciaux et sans accents
- Choisir Authentification SQL Server.
- Choisir un mot de passe qui respecte la stratégie des mots de passe Windows Server



- 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, vérifier que votre connexion est bien créée. Pour cela allez dans l'onglet Sécurité-puis Connexion et repérer votre connexion



Étape 3 : Attribuer les rôles

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

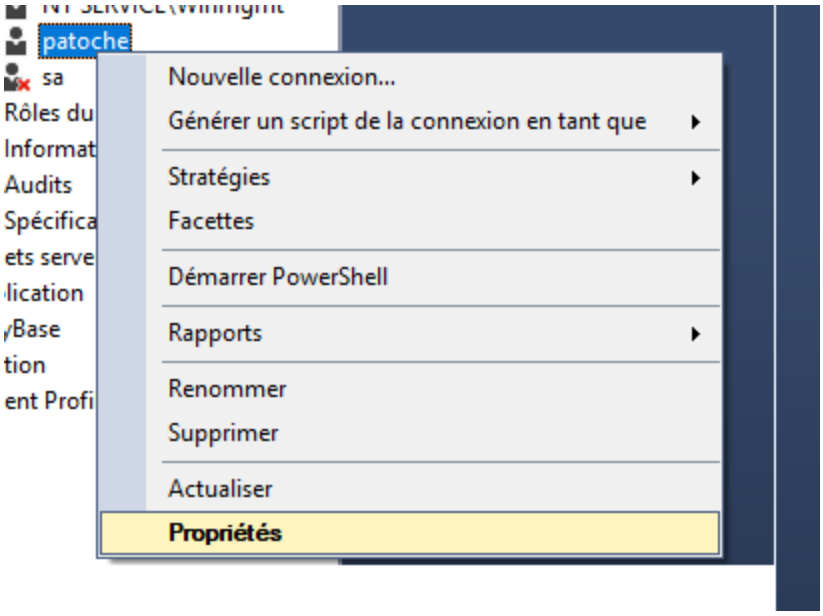
Si vous êtes administrateur alors vous avez déjà ces rôles, sinon vous devez les attribuer à votre connexion avant de créer la bd.

Attention : 

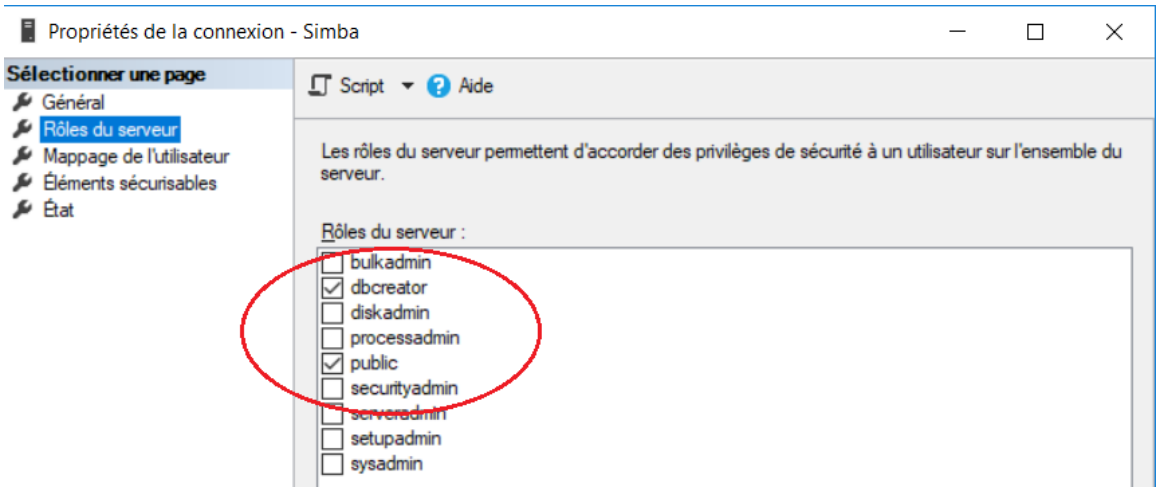
Pour créer la base de données vous devez avoir au moins le rôle **dbcreator**

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.

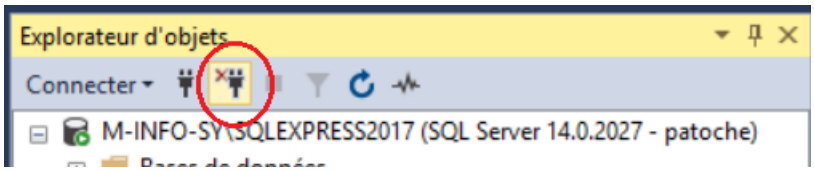
Attention :

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

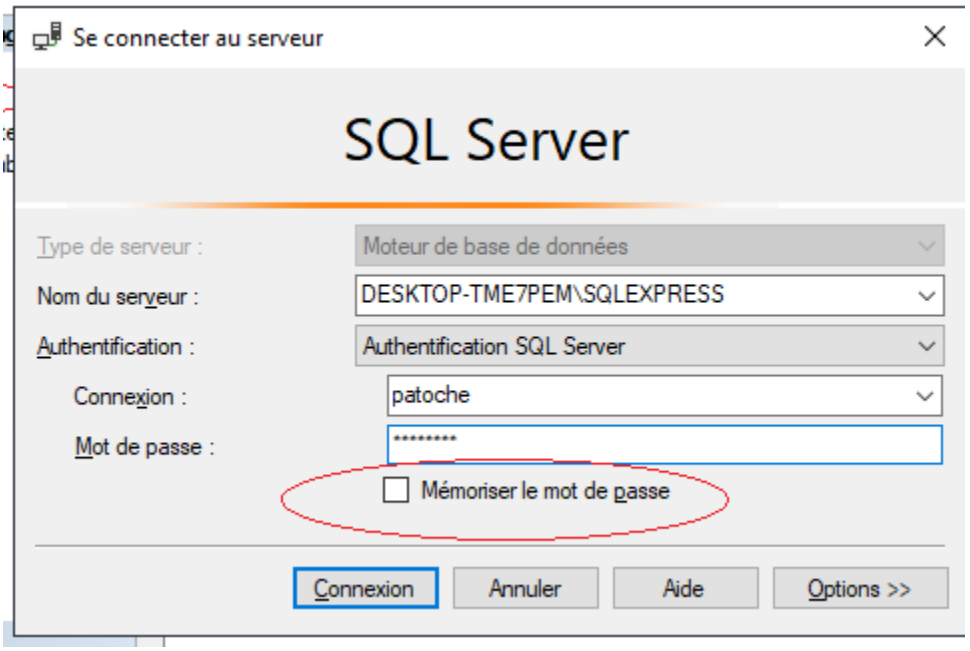
Pour vous déconnecter du serveur, utiliser le bouton Déconnecter



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

Attention :

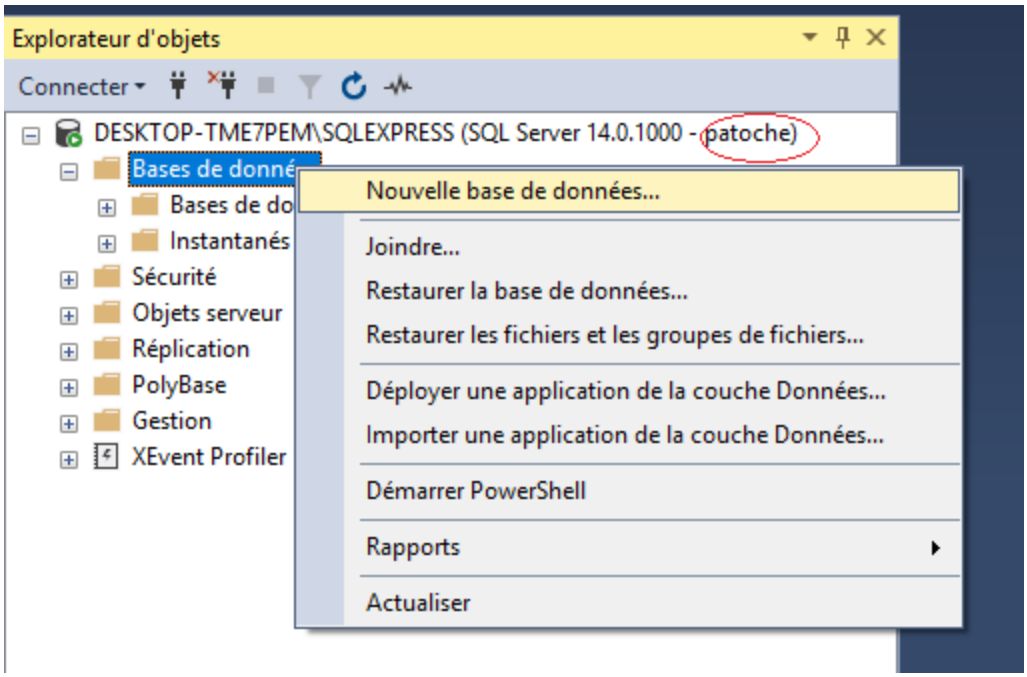
Ne jamais mémoriser le mot de passe



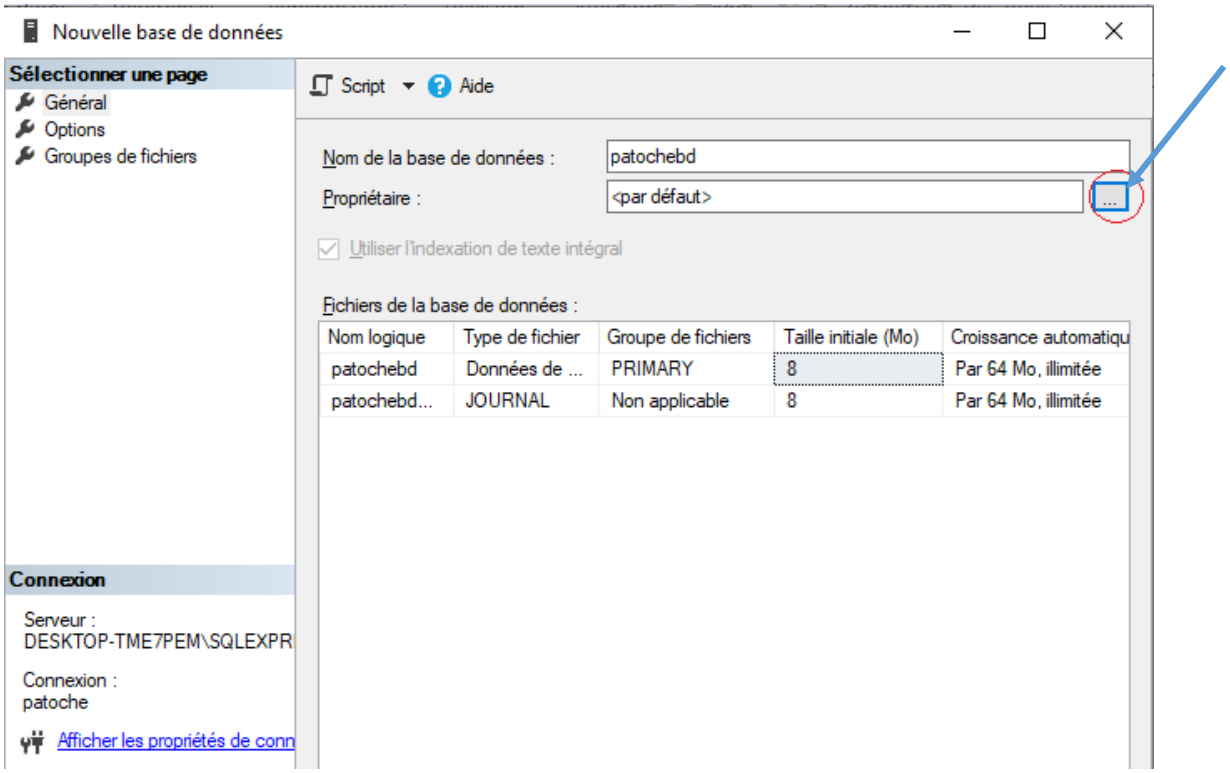
Attention :

Ne jamais mémoriser le mot de passe si vous êtres plusieurs sur une même machine

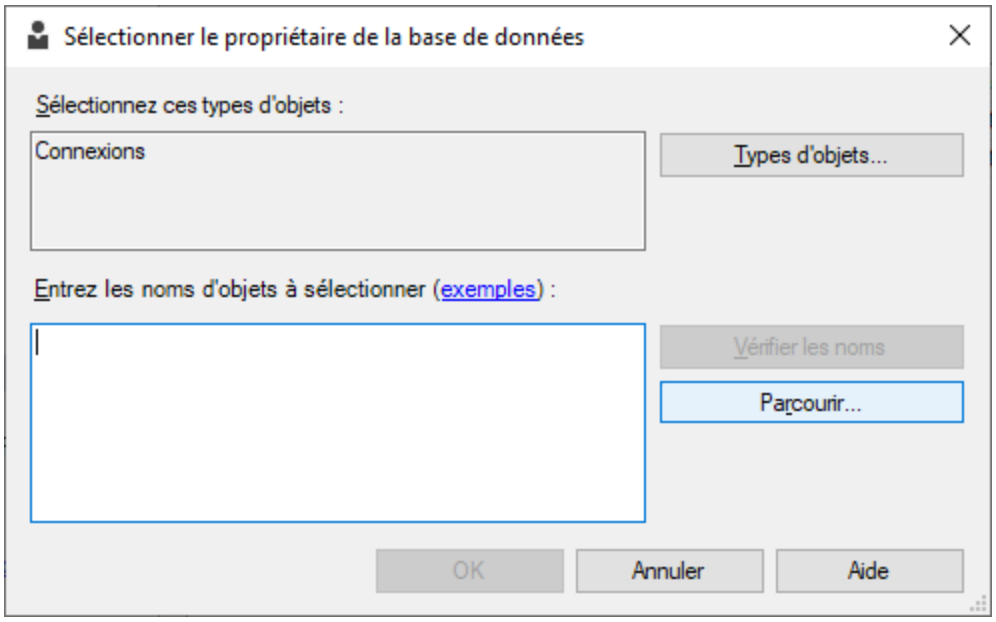
Pour créer une nouvelle base de données, placez-vous à l'onglet bases de données, puis nouvelle base de données. Ou utilise la commande CREATE DATABASE ou vous pouvez le faire pas l'interface comme suit.



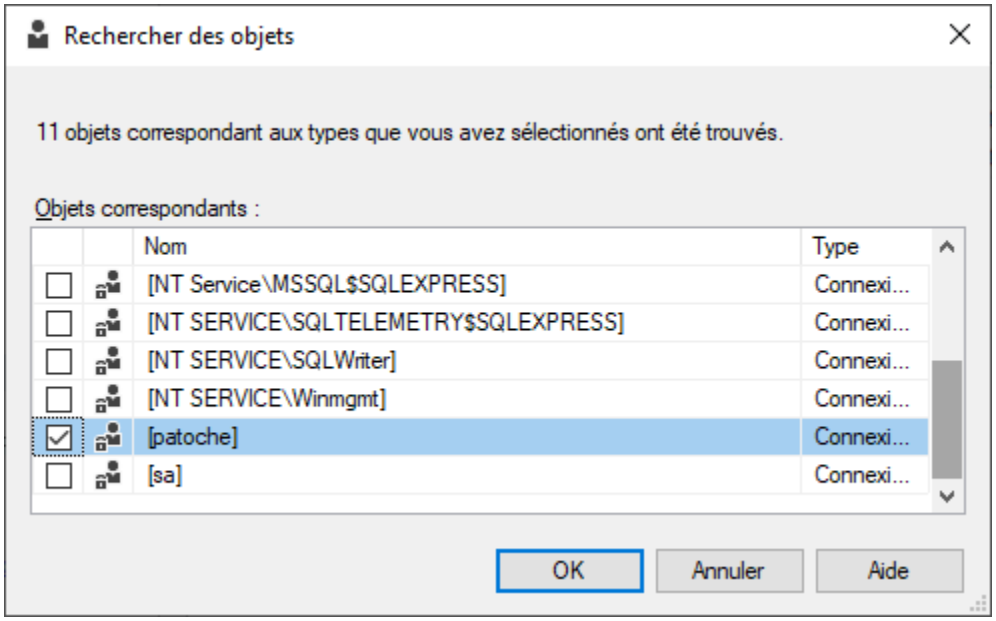
Donnez un nom significatif à votre base de données.



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



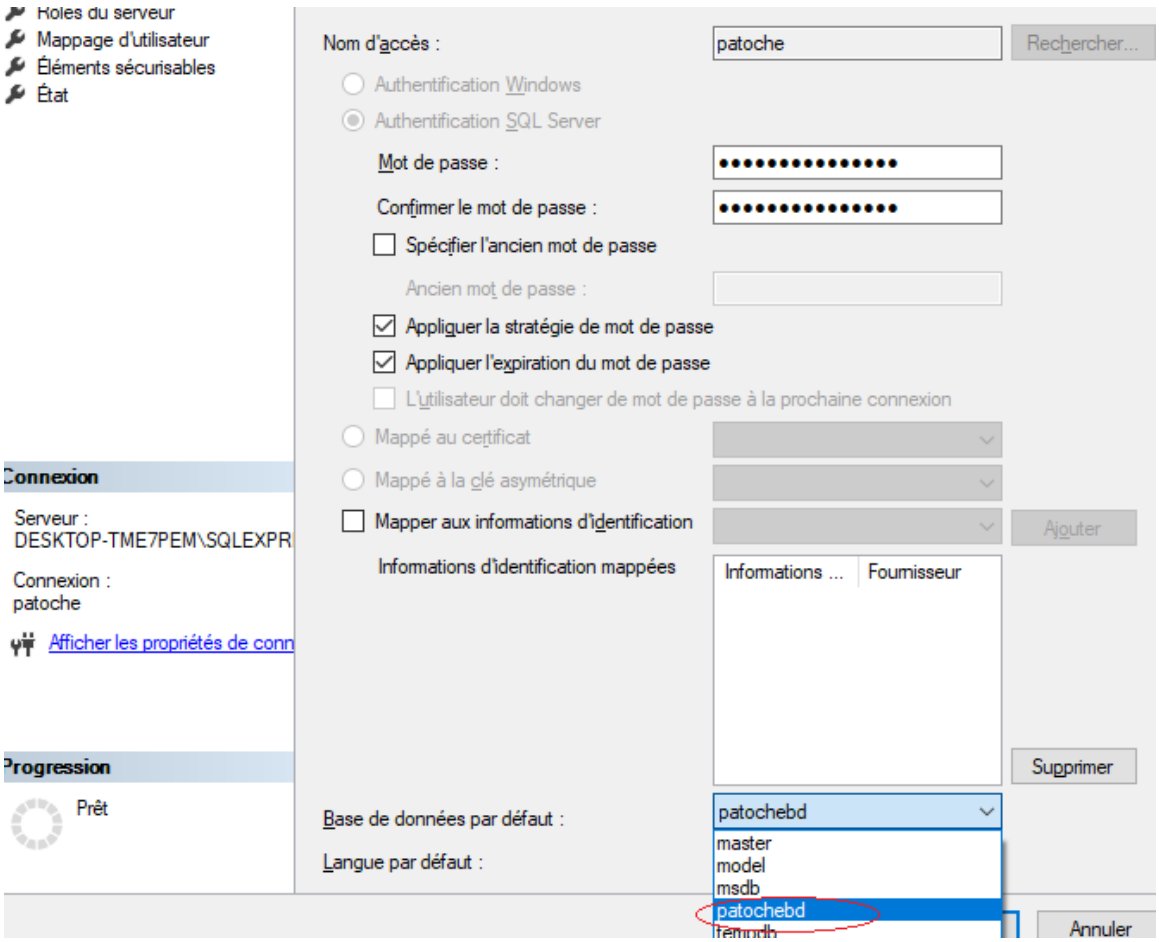
Cliquez ensuite sur parcourir, puis trouvez votre connexion et cochez-la. (Voir figure suivante).



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. **Pour ce faire, il faut se connecter avec « l'authentification Windows » parce que le nouvel utilisateur créé ne possède pas ce droit.**

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



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

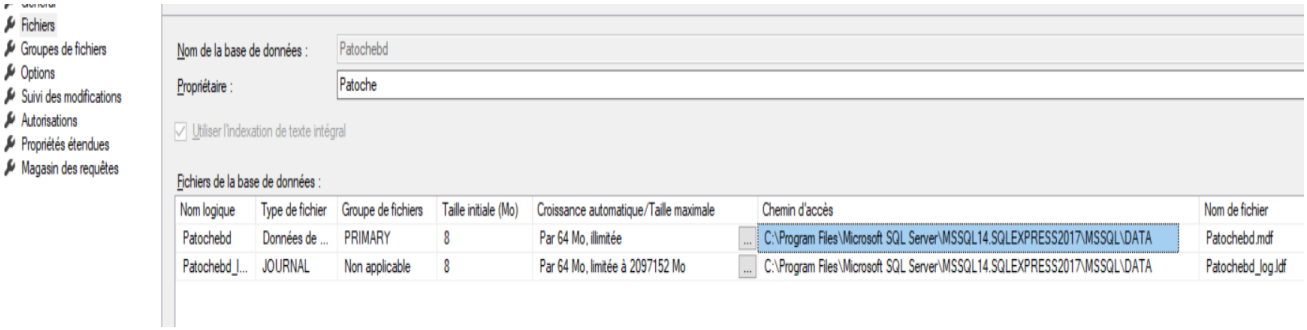
Important :

Vous pouvez également créer votre base de données avec la commande CREATE **et c'est ce qui est recommandé.**

```
CREATE DATABASE nomdelaBD;
```

Où est stockée la base de données ? (ce point sera abordé plus loin).

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.



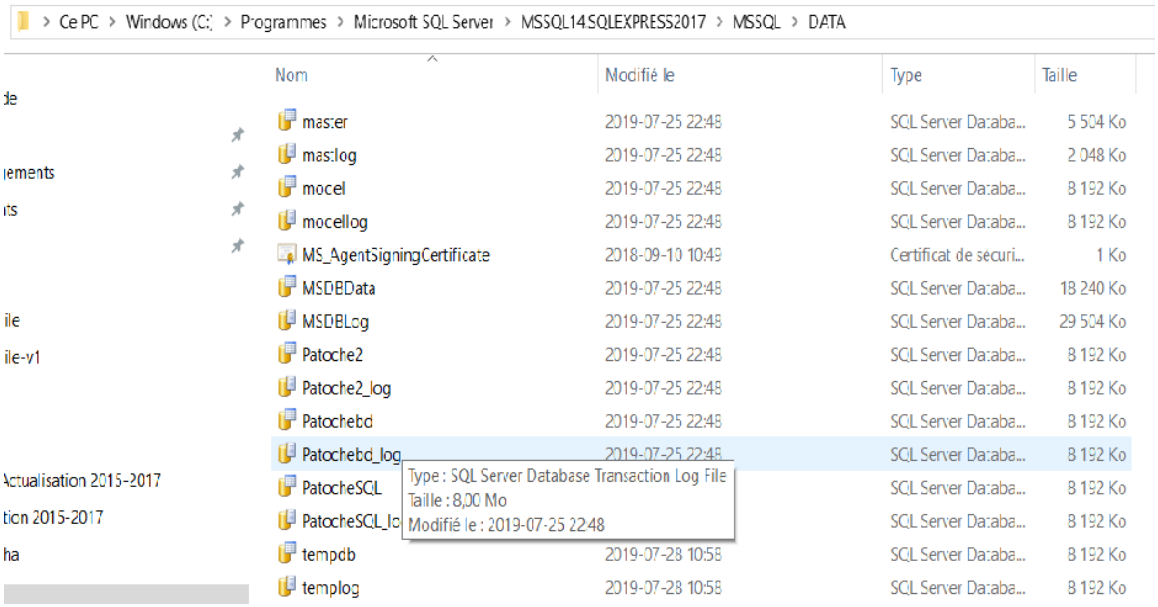
Ces fichiers sont dans :

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

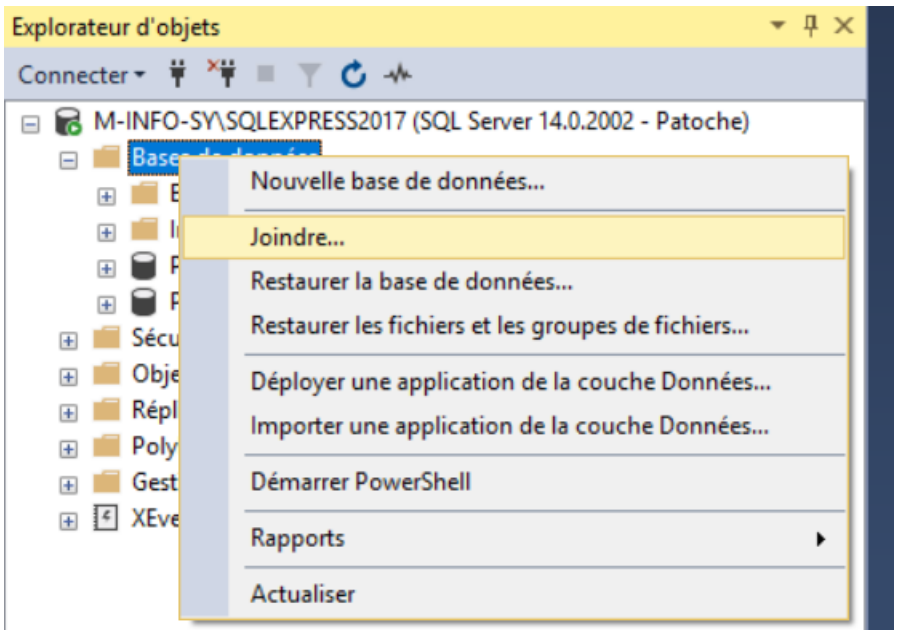
Vous y trouverez deux types de fichiers pour chaque BD : L'un **.mdf** et l'autre. **ldf**

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

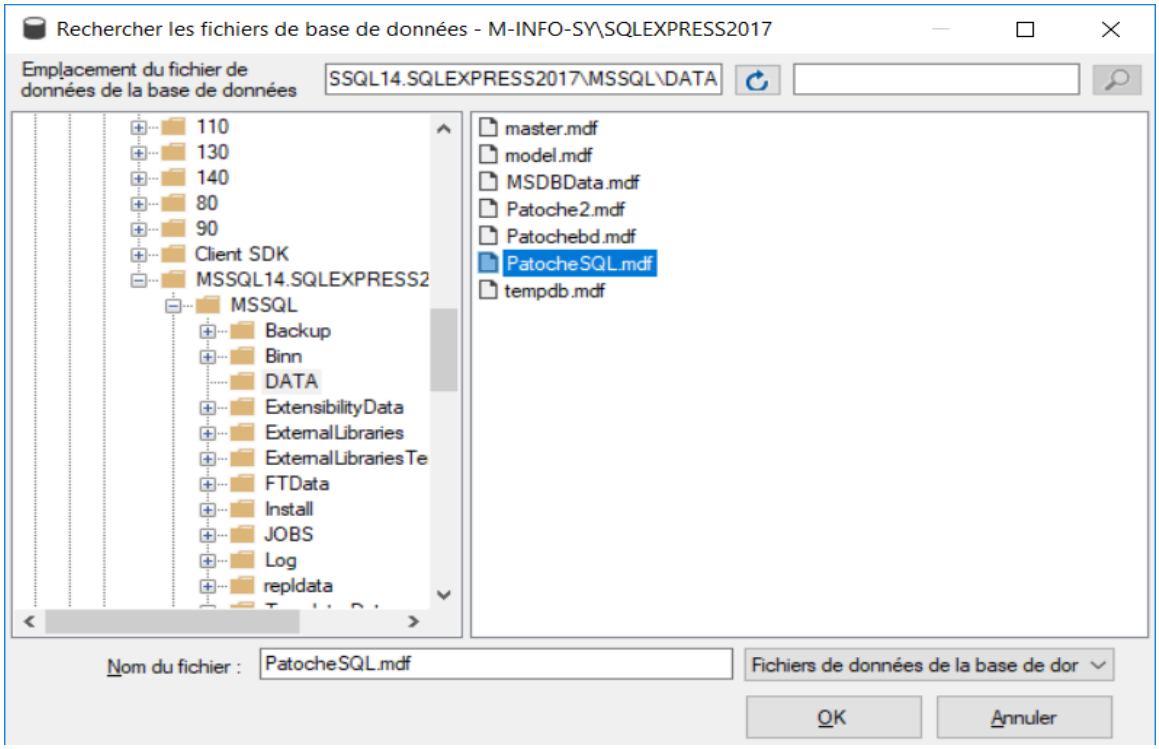


Vous pouvez récupérer votre base en faisant : Bouton droit sur Bases de données, puis **Joindre**. Vous aurez la fenêtre suivante.



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

Votre base de données va apparaitre 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.

Points clés de ce chapitre.

1. MS SQL server est un SGBD Serveur de la compagnie Microsoft.
2. Microsoft SQL Server Management Studio est un des logiciels qui permet d'exploiter une BD MS SQL Server ?
3. On utilise l'authentification Windows pour se connecter la première fois.
4. Avec l'authentification Windows, vous avez tous les droits sur la machine (votre PC) et donc sur le serveur de Bases de données.
5. Pour des raisons de sécurité, il faudra créer des connexions avec des droits restreints et suffisants : Restreints veut dire que vous n'aurez pas accès à tout le serveur. Suffisant veut dire que vous avez assez de droits pour créer et exploiter une base de données sur le serveur.
6. La commande pour créer une base de données est CREATE DATABASE nomBD
7. Pour utiliser une base de de données : USE nomBD
8. Les droits et les rôles seront abordés plus loin (Sécurité)

Chapitre 3, création des tables

Types de données SQL Server

Types numériques exacts

Type	À partir de	À	Stockage
bigint	**-2^63** -9 223 372 036 854 775 808)	**2^63 -1** 9 223 372 036 854 775 807	Huit octets
int	**-2^31** -2 147 483 648)	**2^31 -1** 2 147 483 647	Quatre octets
smallint	**-2^15** -32 768	**2^15 -1** 32 767	Deux octets
tinyint	0	255	1 octets
bit	0	1	Un bit.

Décimal et numériques :

Types de données numériques ayant une précision et une échelle fixes. Les types décimal et numeric sont synonymes et peuvent être utilisés indifféremment.

decimal[**(** *p*[**,** *s*] **)**] et **numeric**[**(** *p*[**,** *s*] **)**]
Valeurs de précision et d'échelle fixes. Lorsque la précision maximale est utilisée, les valeurs valides sont comprises entre - 10^38 +1 et 10^38 - 1.

p (précision) : Nombre total maximal de chiffres décimaux à stocker. Ce nombre inclut le côté à gauche et le côté à droite de la virgule. La précision doit être une valeur comprise entre 1 et la précision maximale de 38. La précision par défaut est 18.

s (échelle) : Nombre de chiffres décimaux stockés à droite de la virgule. Ce nombre est soustrait de *p* afin de déterminer le nombre maximal de chiffres à gauche de la virgule décimale

Money et smallmoney

Type	À partir de	À	Stockage
money	- 922 337 203 685 477.5808	922 337 203 685 477,5807	Huit octets
smallmoney	-214 748.3648	214 748.3647	Quatre octets

Numériques approximatifs

Types de données approximatives à utiliser avec des données numériques à virgule flottante. Les données à virgule flottante sont approximatives ; il n'est donc pas possible de représenter précisément toutes les valeurs de ce type de données

Type	Plage	Stockage
float [**(** *n* **)**]	1,79E+308 à -2,23E-308, 0 et 2,23E-308 à 1,79E+308	Dépend de n
reel	- 3,40E + 38 à -1,18E - 38, 0 et 1,18E - 38 à 3,40E + 38	Quatre octets

Le type DATE

Type	Plage
Date : Date Définit une date dans SQL Server.	Du 1er janvier de l'an 1 (de notre ère) au 31 décembre 9999 (du 15 octobre 1582 au 31 décembre 9999 pour Informatica)
Datetime : Définit une date qui est associée à une heure de la journée avec des fractions de seconde qui se présente au format 24 heures.	Plage de date : Du 1er janvier 1753 au 31 décembre 9999 Plage temporelle 00:00:00 à 23:59:59.997
Time : Définit une heure d'un jour. L'heure ne prend pas en charge les fuseaux horaires et se présente au format 24 heures.	Plage : 00:00:00.0000000 à 23:59:59.9999999 hh comprend deux chiffres, entre 0 et 23, qui représentent l'heure. mm comprend deux chiffres, entre 0 et 59, qui représentent la minute. ss comprend deux chiffres, entre 0 et 59, qui représentent la seconde. n* comprend entre zéro et sept chiffres, entre 0 et 9999999, qui représentent les fractions de seconde.

Chaînes de caractères

Type	Description
Char(n)	Données de type chaîne de taille fixe. *n* définit la taille de chaîne en octets et doit être une valeur comprise entre 1 et 8 000.
Varchar(n|max)	Données de type chaîne de taille variable. Utilisez *n* pour définir la taille de chaîne en octets et peut être une valeur comprise entre 1 et 8 000 ou utilisez **max** pour indiquer une taille de contrainte de colonne maximale de 2^31-1 octets (2 Go).
text	Données non-Unicode de longueur variable figurant dans la page de codes du serveur et ne pouvant pas dépasser en longueur 2^31 - 1 octets (2 147 483 647)

Les chaînes de caractères Unicode

Type	Description
Nchar(n)	Données de type chaîne de taille fixe. *n* définit la taille de chaîne en paires d'octets et doit être une valeur comprise entre 1 et 4 000. La taille de stockage est le double de *n* octets.
Nvarchar(n|max)	Données de type chaîne de taille variable. *n* définit la taille de chaîne en paires d'octets et peut être une valeur comprise entre 1 et 4 000. **max** indique que la taille de stockage maximale est de 2^30-1 caractères (2 Go). La taille de stockage est deux fois *n* octets + 2 octets.
ntext	Données Unicode de longueur variable ne pouvant pas dépasser 2^30 - 1 octets (c'est-à-dire 1 073 741 823). La taille de stockage, en octets, est le double de la longueur de chaîne entrée.

Binary et varbinary

Type	Description
Binary(n)	Données binaires de longueur fixe de *n* octets, où *n* est une valeur comprise entre 1 et 8 000. La taille de stockage est égale à *n* octets.
Varbinary(n|max)	Données binaires de longueur variable. *n* peut être une valeur comprise entre 1 et 8 000. **max** indique que la taille de stockage maximale est de 2^31-1 octets.
image	Données binaires de longueur variable occupant de 0 à 2^31 - 1 (2 147 483 647) octets.

Pour plus de détails, allez sur :

https://learn.microsoft.com/fr-ca/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver16

La propriété « IDENTITY » d'une table

Définition et exemples

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 définie 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 prorité IDENTITY.

Exemple1 :

```
CREATE TABLE Eleves
(
num INT IDENTITY,
nom VARCHAR(30),
prenom VARCHAR(30),
CONSTRAINT pk_eleve PRIMARY KEY(num)
);

INSERT INTO Eleves (nom,prenom) VALUES('Patoche','Alain');
INSERT INTO Eleves (nom,prenom) VALUES('Simba','Chat');
INSERT INTO Eleves (nom,prenom) VALUES('Leroy','Lion');
INSERT INTO Eleves (nom,prenom) VALUES('Simpson','Bart');
```

Nous avons le résultat suivant lorsque nous faisons `SELECT * FROM Eleves;`

Résultats | Messages

	num	nom	prenom
1	1	Patoche	Alain
2	2	Simba	Chat
3	3	Leroy	Lion
4	4	Simpson	Bart

Que remarquons nous ?

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.
4. Par défaut, IDENTITY commence à 1 et l'incrément est 1

Le prochain numéro à insérer est le 5

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
(
idClient SMALLINT IDENTITY(10,2),
nomClient VARCHAR(30),
CONSTRAINT pk_client PRIMARY KEY(idClient)
);


INSERT INTO Clientsinfo (nomClient) VALUES('Gavroche');
INSERT INTO Clientsinfo (nomClient) VALUES('Le fou');
INSERT INTO Clientsinfo (nomClient) VALUES('Le rigolot');
```

Lorsque nous faisons `SELECT * FROM ClientsInfo` , on obtient :

	idClient	nomClient
1	10	Gavroche
2	12	Le fou
3	14	Le rigolot

En principe, le prochain numéro à insérer est le 16.

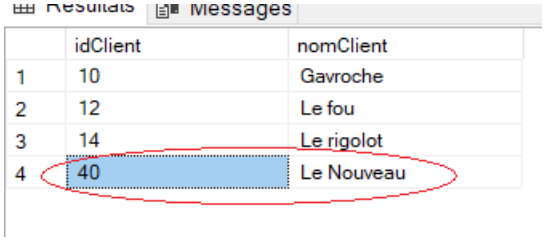
Si on veut faire une insertion **manuelle** dans la colonne idClient (IDENTITY) il faut mettre IDENTITY_INSERT a ON

Exemple 3 :

```
SET IDENTITY_INSERT Clientsinfo ON;

INSERT INTO  ClientsInfo (IdClient,nomClient) VALUES(40,'Le Nouveau');
```

Lorsque nous faisons `SELECT * FROM ClientsInfo` , on obtient :



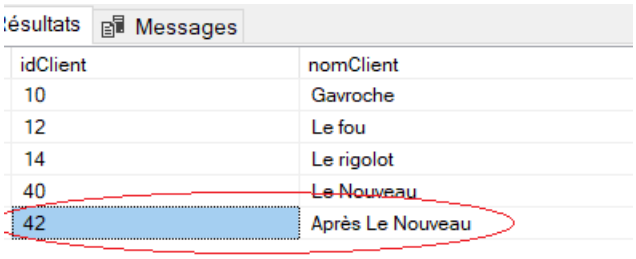
	idClient	nomClient
1	10	Gavroche
2	12	Le fou
3	14	Le rigolot
4	40	Le Nouveau

Question : Si je remets IDENTITY_INSERT pour Clientsinfo à OFF quel est le prochain numéro inséré ? est-ce 42 ou 16 ?; La réponse est **42**.

```
SET IDENTITY_INSERT Clientsinfo OFF;

INSERT INTO  ClientsInfo (nomClient) VALUES('Après Le Nouveau');
```

Lorsque nous faisons `SELECT * FROM ClientsInfo` , on obtient :



idClient	nomClient
10	Gavroche
12	Le fou
14	Le rigolot
40	Le Nouveau
42	Après Le Nouveau

Attention ! Le idClient est rendu à 42, le suivant sera 44, puis 46 et ainsi de suite.

Question : Que deviennent les numéro 16 à 38 ? comment peut-on les insérer ?

Réponse : Ils sont perdus. Si on veut les insérer il faudra le faire manuellement par **SET IDENTITY_INSERT ON**

Récupérer le dernier numéro inséré.

`SELECT @@IDENTITY:` Permet de retourner la dernière valeur insérée pour l'ensemble de la base de données durant la session en cours.

`SELECT IDENT_CURRENT ('nomTable')` Permet de retourner le dernier numéro inséré d'une table.

Voici le contenu initial de nos tables citées dans les exemples précedent.

Table **CilentsInfo**

Résultats | Messages

idClient	nomClient
10	Gavroche
12	Le fou
14	Le rigolot
40	Le Nouveau
42	Après Le Nouveau

Table **Eleves**

Résultats | Messages

	num	nom	prenom
1	1	Patoche	Alain
2	2	Simba	Chat
3	3	Leroy	Lion
4	4	Simpson	Bart

1- On exécute :
```
INSERT INTO Eleves (nom,prenom) VALUES('Saturne','Lune');
```
Saturne aura donc le numéro 5.

2- Si on fait : `SELECT @@IDENTITY` **le résultat sera 5**

3- On fait ensuite
```
INSERT INTO  ClientsInfo (nomClient) VALUES('Le suivant');Le suivant a le numéro 44
```

Si on fait : SELECT @@IDENTITY, **le résultat sera 44**

4- Si on fait **:**
```
SELECT IDENT_CURRENT ('Eleves'), ceci retournera le dernier numéro inséré par la colonne IDENTITY de la table Eleves. Donc va retourner 5
```

Attention : 

@@IDENTITY **retourne le dernier numéro inséré par la colonne IDENTITY. Si plusieurs tables ont une colonne IDENTITY il faudra faire attention pour être certain que c'est la réponse que vous voulez.** @@IDENTITY est très utilisé en procédures stockées.

Si vous voulez absolument récupérez le denier inséré pour une colonne IDENTITY d'une table en particulier, utilisez : `IDENT_CURRENT ('nomTable')`

Création et modification de tables avec SQL Server

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

```
CREATE TABLE Departements
(codeDep CHAR(3),
nomDepartement VARCHAR(30),
CONSTRAINT pk_departement PRIMARY KEY(codeDep)
);

CREATE TABLE Employes
(
empno INT IDENTITY ,
nom VARCHAR(20) NOT NULL,
prenom VARCHAR(30),
salaire MONEY,
codeDep CHAR(3),
CONSTRAINT fk_departement FOREIGN KEY(codeDep)
REFERENCES Departements(codeDep),
CONSTRAINT pk_employe PRIMARY KEY(empno)
);
```

1- La commande CREATE TABLE est la même qu'avec ORACLE. Il faut juste vérifier les types de variables.
2- Les contraintes se définissent de la même façon qu'avec ORACLE
3- La commande ALTER TABLE se définie de la même façon qu'avec ORACLE
4- La commande DROP Table se définie de la même façon qu'avec ORACLE, mais vous n'avez pas l'option CASCADE CONSTRAINTS. Ce qui veut dire que les tables référencées seront détruites en dernier. Vous pouvez ajouter l'option IF EXISTS, dans ce cas si la table n'existe pas, vous n'avez pas de message erreur.

```
ALTER TABLE Employes ADD adresse Varchar(50) NOT NULL;

ALTER TABLE Employes ADD CONSTRAINT ck_salaire CHECK
(salaire>20000);
```

```
DROP TABLE Employes;-→ En 1
DROP TABLE Departements;→ En 2


DROP TABLE  IF EXISTS EmployesClg;
```

Les commandes DML et les transactions

1- Les commandes DML on la même syntaxe qu'avec ORACLE.
2- SQL Server permet de faire des insertions multiples comme suit :

```
INSERT INTO Departements VALUES
('inf','Informatique'),
('rsh','Ressources humaines'),
('ges','Gestion'),
('rec','Recherche et developpement');
```

Pour exécuter un COMMIT après des opérations DML vous avez besoin d'un BEGIN TRANSACTION

```
BEGIN TRANSACTION;
INSERT INTO EMPLOYES (nom, prenom,salaire,codeDep,adresse) VALUES
 ('Patoche','Alain',22000.5,'ges', '11 rue de la lune'),
 ('Gavroche','Miserable',65000,'inf', '12 rue de jupiter'),
 ('Bien','Henry',56000,'inf','Ici saturne'),
 ('Leriche','Alain',400000.00,'rec','Sur mars');
COMMIT;
```

La commande SELECT

La commande SELECT a la même syntaxe que dans ORACLE.

Pour extraire le N première Lignes, MS SQL Server utilise TOP tandis que Oracle utilise le ROWNUM avec une sous requête.

Exemple :Pour chercher les deux employés avec le salaire élevé

Avec ORACLE

```
SELECT * FROM (
SELECT nom, prrenom , salaire
FROM Employes
```

```
ORDER BY salaire DESC)
WHERE ROWNUM <=2;
```

Avec MS SQL Server

```
SELECT TOP 2 nom, prenom, salaire
FROM EMPLOYES ORDER BY salaire DESC;
```

Points clés du Chapitre

1- Les commandes DDL (Data Defintion Language) sont les même qu'avec Oracle que vous avez vu en session2. Seul les types de données changent. Dans la commande DROP TABLE vous n'avez pas CASCADE CONSTRAINTS
2- MS SQL Server permet de faire une auto incrémentation de la clé primaire avec la propriété IDENTITY
3- Les commandes DML (Data Manipulation Language) sont les mêmes qu'avec Oracle.
4- Les Transactions : Pour faire un COMMIT vous avez besoin de d'avoir un BEGIN TRANSACTION. Le ROLLBACK se fait jusqu'au dernier BEGIN TRANSACTION.
5- Il y a un chapitre sur les transactions plus loin.
6- La Commande SELECT se fait de la même façon qu'avec ORACLE. Les jointures se font avec INNER JOIN (ou RIGHT /LEFT OUTER JOIN)
7- Pour afficher les N première lignes utiliser SELECT **TOP N**

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.

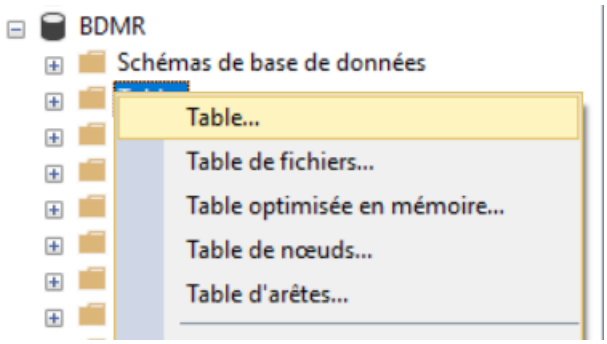
Vous pouvez soit obtenir un modèle relationnel d'une base de données déjà créée ou tout simplement créer un nouveau schéma.

Étape 0 : création de la base de données

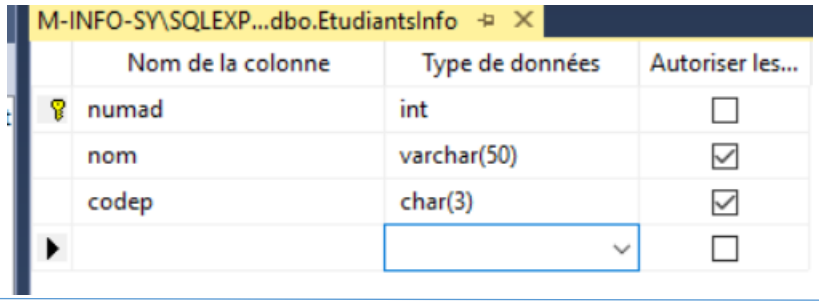
Créer une nouvelle base de données avec un nom significatif (ou votre nom) Faîtes en sorte que vous en soyez le propriétaire.

Étape 2 : Création des tables :

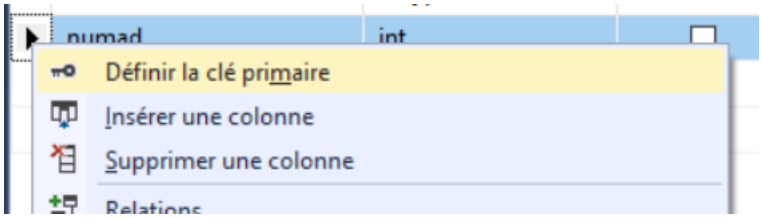
1- Faire bouton droit de sur l'onglet Tables de votre BD, puis table



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

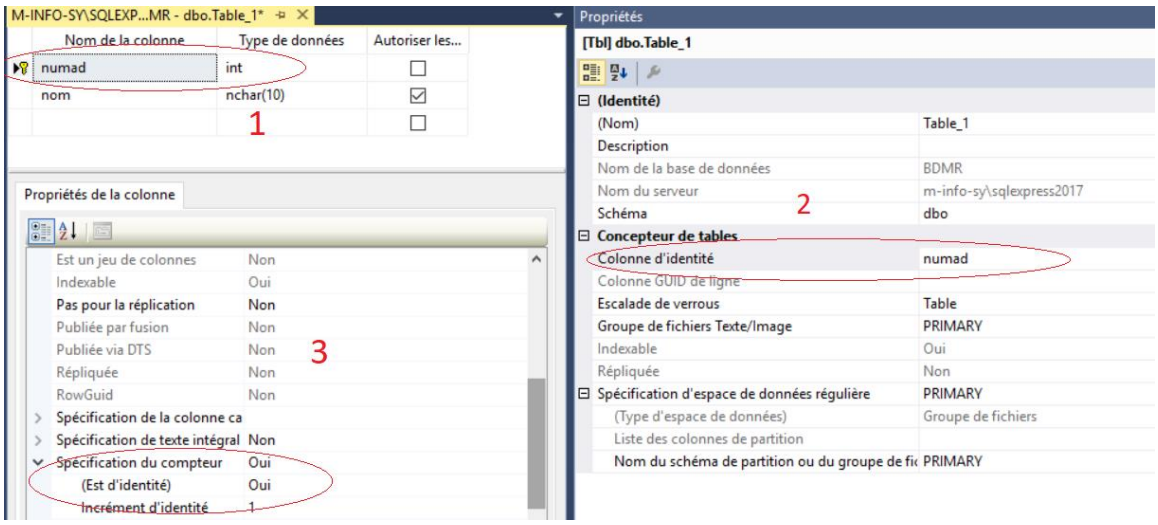


3- Sélectionner la colonne de vous voulez qu'elle soit clé primaire, puis bouton droit et faire : définir la clé primaire : cette étape est obligatoire si vous voulez que la BD soit en 1FN.



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

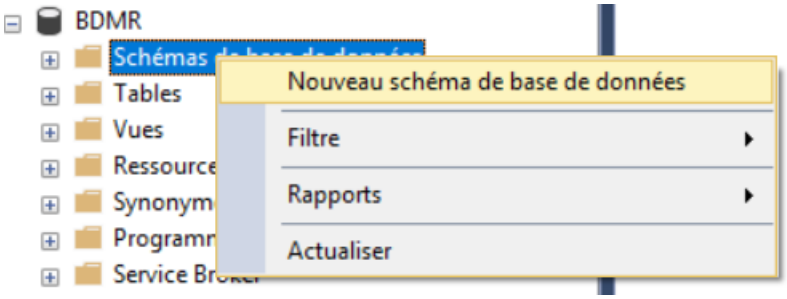


4- Enregistrer la table : cliquez sur le bouton enregistrez, puis donnez un nom à votre table. Notre table a pour nom : **EtudiantsInfo**

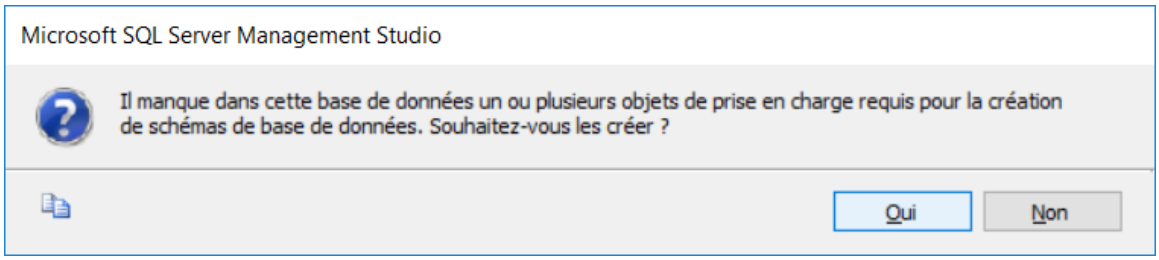
Étape 3, créer le schéma de la BD

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

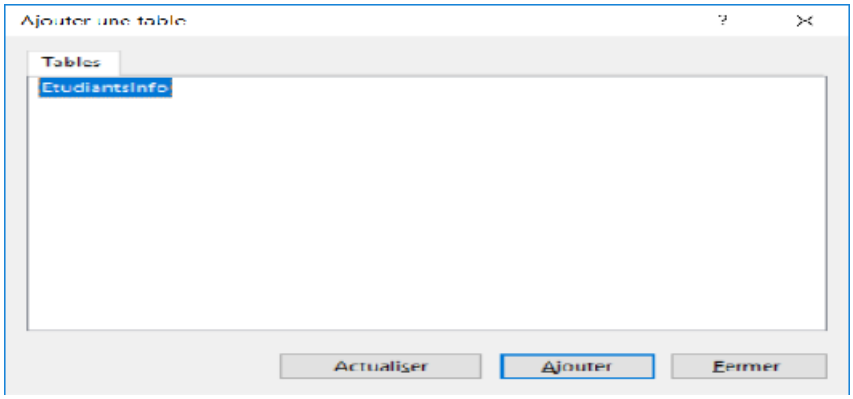
1. Sélectionner Diagrammes de base de données, sur le bouton droit de la souris, faire nouveau schéma :



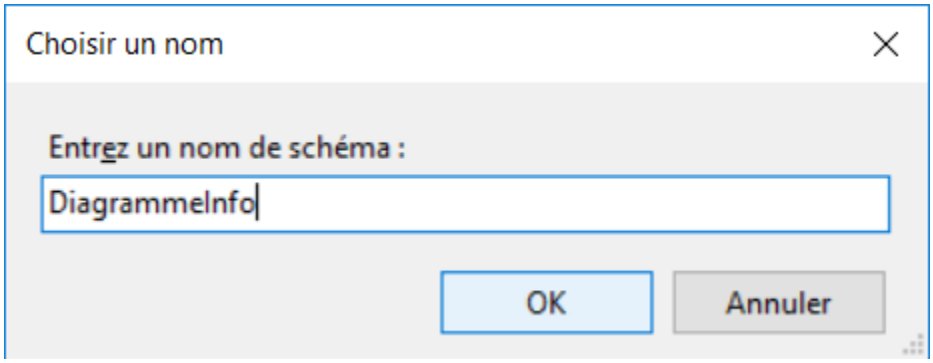
2. Si vous avez ce message, faites OK



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



4. Enregistrez votre diagramme.

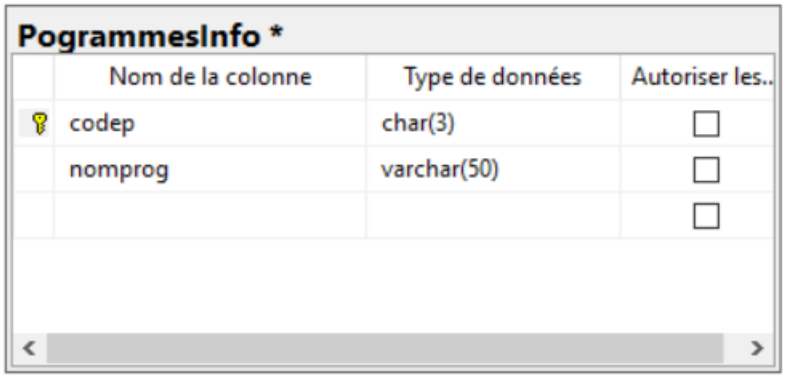


Étape 4 : Définir les relations (la clé étrangère)

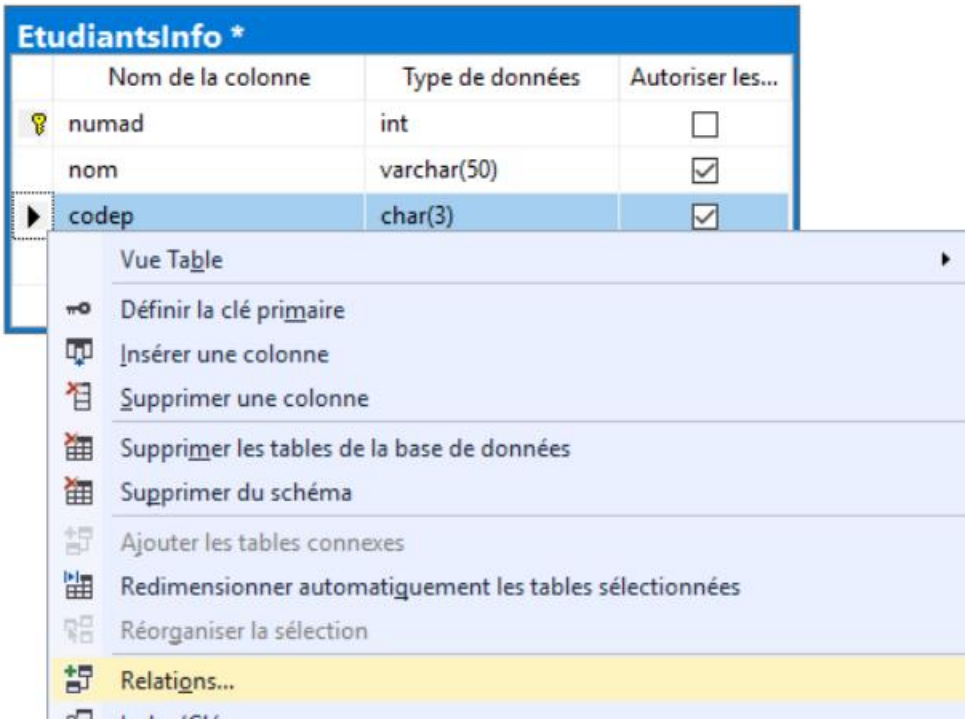
Une fois que vos tables sont créées, ou bien au fur et à mesure que vos tables vont se créer, il sera important de définir les liens entre les tables. Ces liens sont évidemment définis par le concept de Foreign Key ou clé étrangère.

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.

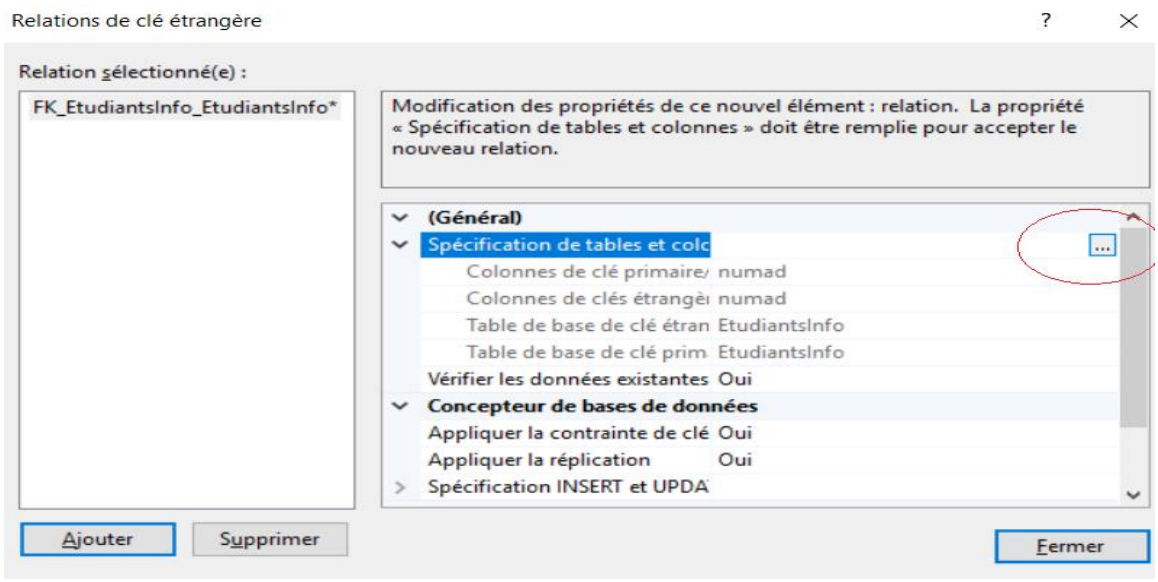
On suppose que la table ProgrammesInfo est créée.



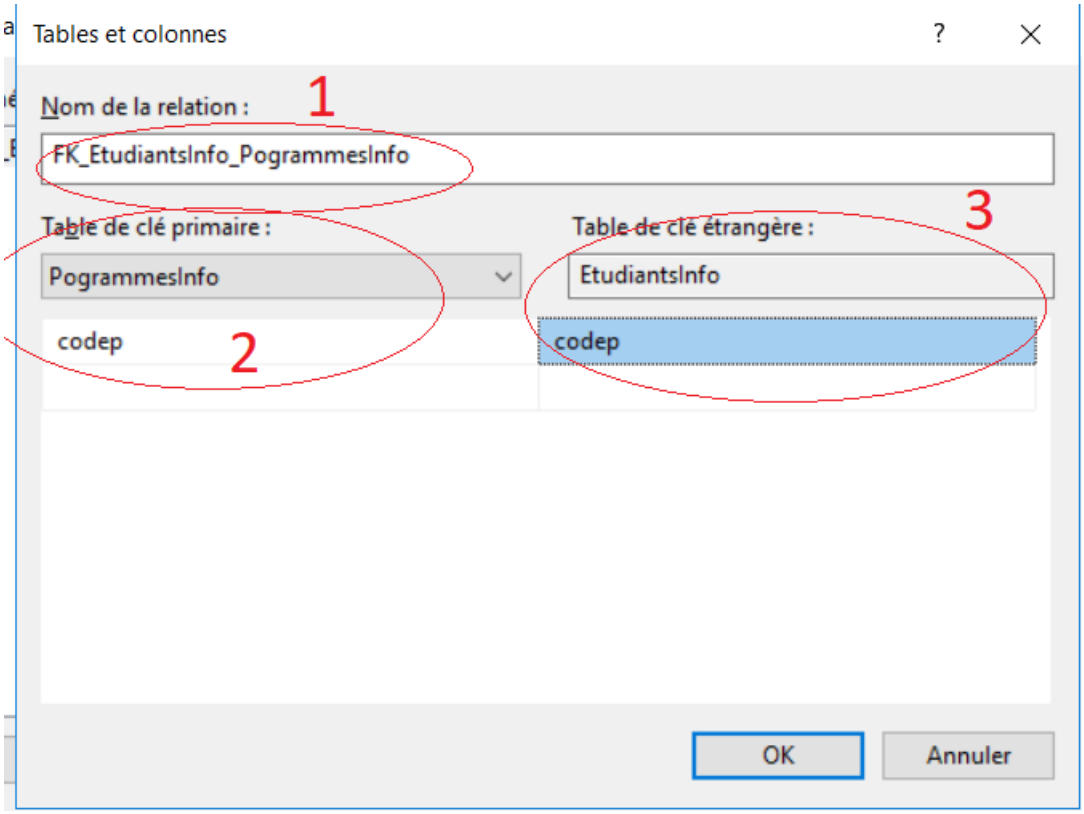
1. Sur la colonne Codep de EtudiantsInfo, (la colonne qui sera clé étrangère), faire Relation comme le montre la figure



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



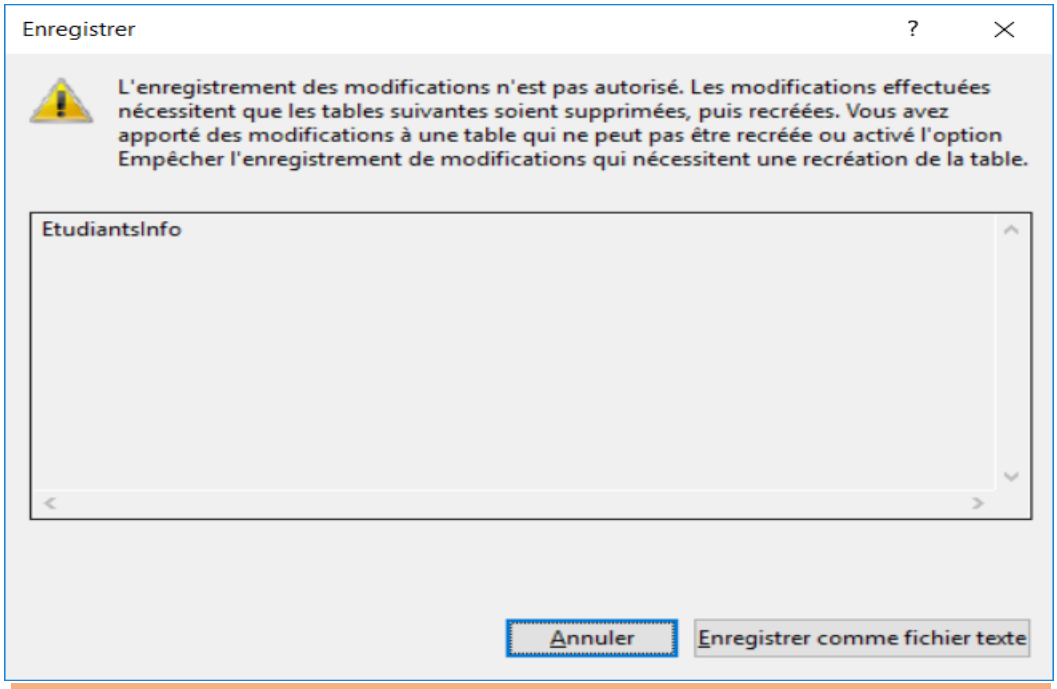
4. Vérifiez que vous avez bel et bien les bonnes colonnes avec les bonnes tables :
    a. Vous pouvez changer le nom de la contrainte de FK →1
    b. Vérifier la table et la colonne de la primary Key →2
    c. Vérifier la table et la colonne de la FK→3



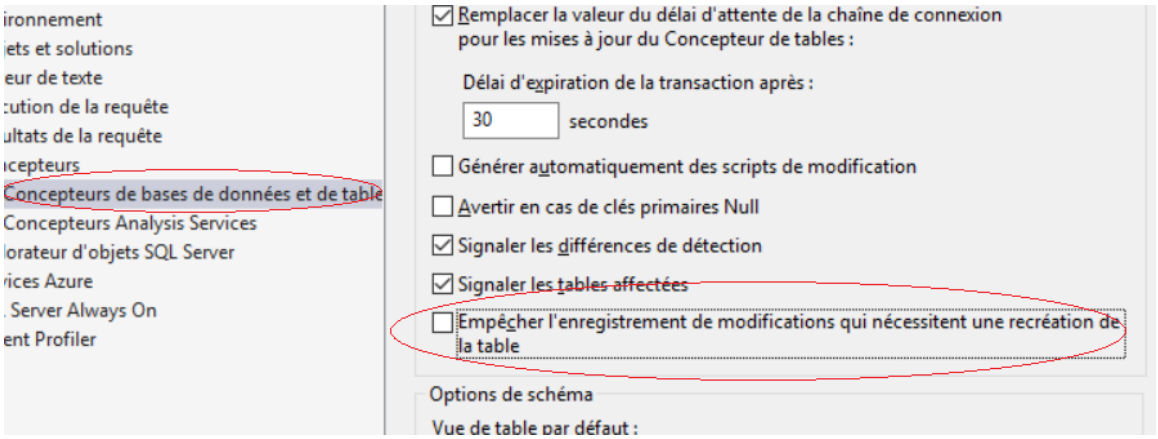
5. Faire OK, puis fermer pour terminer.
6. Enregistrez.

Attention : 

Si au moment d'enregister 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



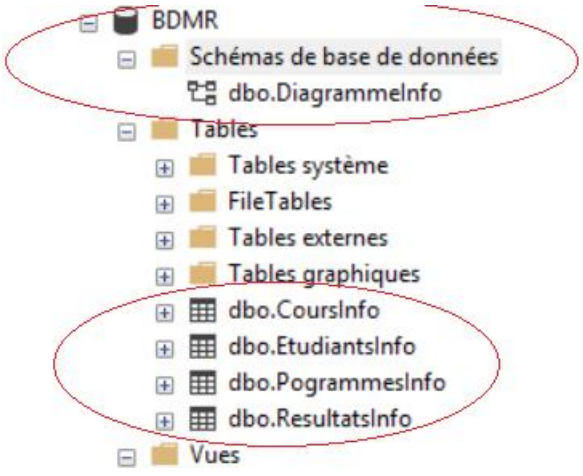
Définir la clé primaire composée

Pour définir une clé primaire composée sur une table, c'est très facile. Il suffit des 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

votre 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.

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;
```

Affectation de valeur aux variables : à l'initialisation de la variable

```
DECLARE @variable  int =12;
PRINT @variable;


DECLARE @nom varchar(30) ='Patoche'
PRINT @nom;
```

Lorsqu'on veut déclarer plusieurs variables en même temps, un seul DECLARE est suffisant. Il faudra terminer les instructions de déclaration (sauf la dernière) par des virgules.

```
DECLARE
@variable  INT =12,
@nom VARCHAR(30) ='Patoche',
@date DATE ='2020-08-21';

PRINT @variable; PRINT @nom; PRINT @date;
```

Affectation de valeur aux variables : avec le mot réservé SET

```
DECLARE @salaire MONEY;
SET @salaire = 45000
PRINT @salaire;
```

```
DECLARE
@variable  INT =12,
@nom VARCHAR(30),
@date DATE ;
SET @nom ='Patoche';
SET @date ='2020-09-30';

PRINT @variable; PRINT @nom; PRINT @date;
```

Affectation de valeur aux variables : par des valeurs provenant d'une table (SELECT)

En générale, nous avons besoin d'extraire des données de la base de données et de les affecter à des variables afin de pouvoir les tester, les traiter etc.. dans ce cas on utilise la commande SELECT pour extraire une donnée puis l'affecter à une variable.

```
USE bdEmployesClg
DECLARE
@salaire_min MONEY
BEGIN
SELECT @salaire_min =MIN(SALAIRE) FROM Employes;
PRINT @salaire_min;
END;
```

Les mots réservés : BEGIN …END

Ces mots réservés permettent de définir un bloc ou un groupe d'instructions qui doivent être exécutées. Un peu comme { } en C#.

Les structures de contrôles :

L'alternative :

L' Instruction IF: syntaxe

```
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 } ]
```

Exemple1: (ce bout de code est ce que nous appelons un bloc anonyme)

```
DECLARE
@vsalaire MONEY;
BEGIN
SELECT  @vsalaire =AVG(SALAIRE) FROM Employes WHERE codeDep ='inf';
     IF (@vsalaire>60000) UPDATE Employes SET Salaire = Salaire +
     0.01*salaire  WHERE codeDep ='inf';

     ELSE UPDATE  Employes SET Salaire = Salaire + 0.02*salaire
       WHERE codeDep  ='inf';
END;
```

Dans l'exemple précèdent :

1- On déclare une variable @vsalaire de type MONEY
2- Le bloc d'instructions SQL commence par BEGIN
3- On cherche le salaire moyen des employés du département 'inf' puis on l'affecte à la variable @vsalaire.

4- On vérifie : si le salaire moyen des employés du département ‘inf’ est plus élevé que 60000 on augmente le salaire des employés de ce département de 1% sinon on l’augmente de 2%
5- On termine le bloc d’instructions par END.

Attention: 

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

D’autres exemples plus loin avec la notion de curseur

Exemple 2, le IF et le ELSE comporte plus d’une instruction

```
begin
declare @cat char(1);
set @cat ='A'
if(@cat='S')
     begin
     select enonce from questions where idCategorie =@cat;
     select 1;
     end;

else
     begin
     select nomCategorie from categories;
     select 2
     end;
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 pour un SELECT

```
SELECT   nom,prenom, codeDep =
      CASE  codeDep
         WHEN 'inf' THEN 'Informatique'
         WHEN 'rsh' THEN 'Ressources humaines'
         WHEN 'ges' THEN 'Gestion'
          WHEN 'rec' THEN 'Recherche et developpement'
         ELSE 'aucun département connu'
      END, salaire
FROM employes  ;
```

Dans l'exemple précédent, on affiche le nom des départements selon (CASE) leur code (codeDep). Pour un affichage probablement que nous n'avons pas besoin de faire une jointure pour aller chercher les noms des départements dans la table Departement. Un CASE est suffisant.

Exemple 2, CASE pour UPDATE

```
UPDATE employes SET salaire =
    ( CASE
         WHEN (salaire < 25000 ) THEN salaire + 0.05*salaire
         ELSE (salaire + 0.01*salaire)
       END
    ) ;
```

Dans l'exemple précédent, la mise à jour du salaire dépend de sa valeur initiale. Dans ce cas un CASE et approprié.

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 150000. Mais si le maximum des salaires dépasse 500 000, on arrête,

```
BEGIN
     WHILE (SELECT  AVG (salaire) FROM employes )<= 150000
     BEGIN UPDATE employes SET salaire = salaire +1000;
     IF(SELECT MAX(salaire) FROM employes) >500000 BREAK;
     ELSE CONTINUE;
     END;
END;
```

Le mot réservé : GO

GO ne fait pas partie du SQL. C'est un mot réservé **optionnel** qui permet de finir un lot d'instructions.

Exemple :

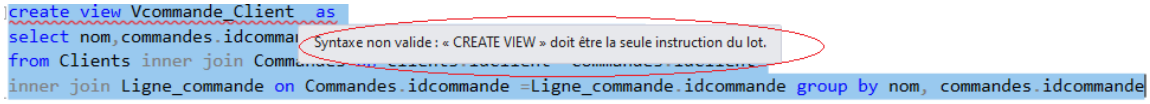
Les procédures et les fonctions stockées sont comme vos fonctions C#. En principe, à moins que ce soit dans un programme principal, vous ne pouvez pas mettre les fonctions et les procédures dans un même fichier.

Si vous finissez votre procédure ou votre fonction par GO dans ce cas, vous pouvez mettre vos procédures et/ou fonctions dans un même fichier sans qu'il y soit des erreurs. Évidement ces procédures s'exécutent séparément.

Autre exemple :

Dans la figure suivante, le message indique, que le CREATE VIEW doit être la première instruction du lot . Même si l'exécution de l'instruction se fait normalement (sans erreur) le Management Studio voit ceci comme une erreur car le lot n'est pas terminé.

Si je rajoute le GO à la fin, je n'aurais plus ce « Warning »..



Avec le GO, je n'ai plus ce message

```
create view Vcommande_Client  as
select nom,commandes.idcommande,sum(Montant) as totalCommande
from Clients inner join Commandes on clients.idClient =Commandes.idClient
inner join Ligne_commande on Commandes.idcommande =Ligne_commande.idcommande
group by nom, commandes.idcommande;
GO;
```

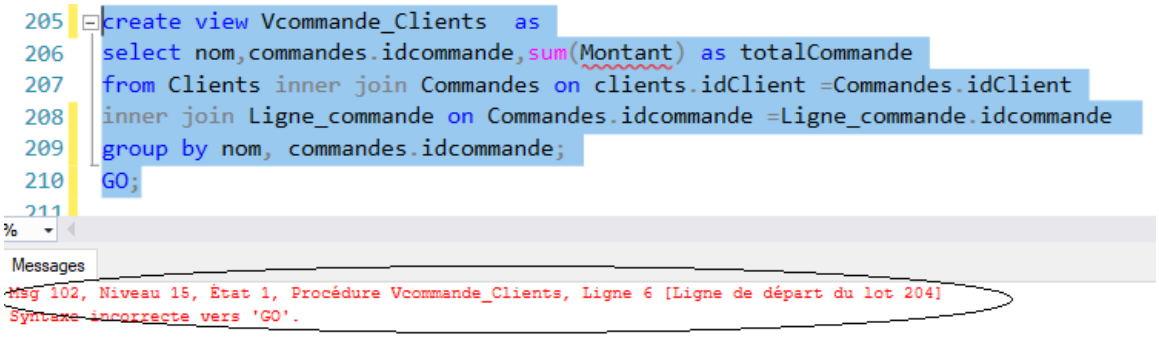
Attention: 

Le GO est vraiment optionnel.

Lorsque vous exécutez votre ou vos instructions du lot, il ne faut pas inclure le GO. Il va renvoyer une erreur. (pourquoi ? Car ce n'est pas du SQL

Mettez le GO pour finir un lot, mais ne l'exécutez-pas SVP

Lorsque le GO est inclus dans l'exécution, il y'a une erreur.



Point Clés du Chapitre :

1- Dans Transact-SQL la déclaration de variable se fait par le mot DECLARE
2- Les variables sont précédées du symbole @
3- Les types de variable sont les type SQL de SQL Server
4- L'affectation de valeurs aux variables se fait par = ou le SET
5- L'affectation de valeurs issues de la base de données à une variable se fait par un SELECT : SELECT @variable = donnée issue de la BD
6- Chaque bloc d'instructions commence par BEGIN et fini par END
7- Le IF a le même rôle de que ce que vous connaissez en C#, cependant il n' a pas la même syntaxe.
8- Le WHILE a le même rôle de que ce que vous connaissez en C#, cependant il n' a pas la même syntaxe.
9- Le GO est optionnel (n'est pas du SQL) et permet de terminer un lot.

Chapitre 6, les transactions

Notions de Transactions :

Une transaction est un bloc d'instructions DML exécuté 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.

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 @@TRANCOUNT de 1. Cette variable système retourne le nombre de BEGIN Transaction exécuté pendant la connexion en cours. Lorsqu'une transaction est comité (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 ou 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.

Exemples :

Soit le code suivant : deux tables. Une table Questions et une table Réponses.

Dans cet exemple, chaque question a 4 réponses dont une bonne réponse.

La table QuestionTrivia a été créée avec la propriété IDENTTY.

```
create database BDExemple;
use BDExemple;

-----
create table CategorieTrivia
(
idCategorie char(1),
nomCategorie varchar(30) not null,
couleur varchar(30) not null,
constraint pk_categorieTrivia primary key (idCategorie)
);

----------------------------------
create table QuestionTrivia
(
idQuestion smallint identity(100,1),
enonce varchar(100) not null,
flag char(1) not null default 'n',
difficulte char(1) not null,
idCategorie char(1) not null,
constraint pk_QuestionTrivia primary key (idQuestion),
constraint ck_flagQuestion check(flag='o' or flag='n'),
constraint ck_difficulte check
                  (difficulte='d' or difficulte='m'or difficulte='f'),
constraint fk_categorieTrivia foreign key (idCategorie)
                  references CategorieTrivia(idCategorie)
);

------------------------------
create table ReponseTrivia
(
idReponse int identity (1,1),
laReponse varchar(100) not null,
estBonne char(1) not null,
idQuestion smallint not null,
constraint pk_reponse primary key(idReponse),
constraint ck_bonne check(estBonne='o' or estBonne ='n'),
constraint fk_question foreign key(idQuestion)
                                    references QuestionTrivia(idQuestion)
);
```

Pour que ce soit cohérent, il faudra saisir les questions ET les réponses de la question en même temps. Donc à l'intérieur d'une transaction. Dans un BEGIN TRANSACTION et UN COMMIT.

Exemple1 : La BD est dans un état cohérent

1- Les tables Questions et réponses sont vides : Aucun enregistrement.
2- SELECT @@TRANCOUNT donne comme **résultat 0** (puisqu'aucune transaction)
3- À la première insertion dans la table Questions, le idQuestion =1;

```
BEGIN TRANSACTION
 INSERT INTO QuestionTrivia(enonce, difficulte, flag) VALUES
 ('De quelle ouleur est le cheval blanc de Napoléon ?','f','n');

 INSERT INTO ReponseTrivia values (1,'Blanc','o',1);
 INSERT INTO ReponseTrivia values (2,'Rouge','n',1);
 INSERT INTO ReponseTrivia values (3,'Noir','n',1);
 INSERT INTO ReponseTrivia values (4,'Jaune','n',1);
 COMMIT ;
```

1- À l'exécution du code aucune erreur n'est renvoyée.
2- BEGIN TRANSACTION augmente @@TRANCOUNT de 1
3- COMMIT diminue le @@TRANCOUNT de 1
4- Le @@TRANCOUNT est donc égal à Zéro. La BD est dans un état cohérent
5- Les table Questions et Réponses contiennent des données cohérentes.

Table Questions

idQuestion	enonce	difficulte	flag
1	De quelle ouleur est le cheval blanc de Napoléon ?	F	N

Table Reponses :

idReponse	enonce	estBonne	idQuestion
1	Blanc	O	1
2	Rouge	N	1
3	Noir	N	1
4	Jaune	N	1

Exemple 2 : la BD n'est pas dans un état cohérent

```
BEGIN TRANSACTION
 INSERT INTO QuestionTrivia(enonce, difficulte, flag) VALUES
 ('De quelle couleur est le mouton noir de Séraphin ?','f','n');

 INSERT INTO ReponseTrivia values (11,'Blanc','n',2);
 INSERT INTO ReponseTrivia values (12,'Rouge','n',2);
 INSERT INTO ReponseTrivia values (12,'Noir','o',2);
 INSERT INTO ReponseTrivia values (14,'Jaune','n',2);
 COMMIT ;
```

L'insertion a été faite dans la table Questions

Résultats Messages

idQuestion	enonce	difficulte	flag
1	De quelle ouleur est le cheval blanc de Napoléon ?	F	N
2	De quelle ouleur est le mouton noir de Séraphin ?	F	N

Mais l'insertion dans la table réponse n'est pas correcte . Nous avons inséré 3 enregistrements au lieu de 4. La clé primaire est dupliquée

idReponse	enonce	estBonne	idQuestion
1	Blanc	O	1
2	Rouge	N	1
3	Noir	N	1
4	Jaune	N	1
11	Blanc	N	2
12	Rouge	N	2
14	Jaune	N	2

Exemple 3 : TRY ---CATCH, ce qui recommandé

Au fond, vous voulez faire l'insertion dans la table Questions uniquement si l'insertion (toutes les opérations DML) dans la table Reponses s'est bien déroulé

Comme la clé primaire est dupliquée dans la table réponses, le système fiat un ROLLBACK pour annuler l'insertion dans la table Questions et dans la table Reponses. De cette façons les données de votre base de données resteront toujours cohérentes

Dans l'exemple suivant, AUCUNE insertion n'est faîtes dans la table Questions, et aucune Question n'est faîte dans la table réponses.

```
BEGIN TRY
      BEGIN TRANSACTION
       INSERT INTO QuestionTrivia(enonce, difficulte, flag) VALUES
      ('Qui a écrit: les neiges de kilimandjaro  ?','M','n');

      INSERT INTO ReponseTrivia values (11,' John Steinbeck','n',2);
      INSERT INTO ReponseTrivia values  (12,'Ernest Hemingway','o',2);
      INSERT INTO ReponseTrivia values  (12,'Victor Hugo','n',2);
      INSERT INTO ReponseTrivia values  (14,'Edgar Frank Codd','n',2);
      COMMIT;

END TRY
BEGIN CATCH
ROLLBACK;
END CATCH
GO;
```

Après corrections (On change le numéro 12 par 13 dans le 3e INSERT INTO) les données s'insèrent proprement.

Autre exemple

On déclare une variable idQ qui va recevoir le dernier numéro de la question que l'on vient d'insérer comme suit : `SELECT @idQ =@@IDENTITY;`

```
BEGIN
 DECLARE @idQ INT;
      BEGIN TRY
            BEGIN TRANSACTION
            INSERT INTO QuestionTrivia(enonce, difficulte, flag) VALUES
            ('Qui a peint: les femmes d''Alger  ?','d','n');
            SELECT @idQ =@@IDENTITY;
            INSERT INTO ReponseTrivia values (21,' Vincent van Gogh','n',@idQ);
            INSERT INTO ReponseTrivia values (22,'Léonard de Vinci','n',@idQ);
            INSERT INTO ReponseTrivia values (23,'Claude Monet','n',@idQ);
            INSERT INTO ReponseTrivia values (24,'Picasso','o',@idQ);
            COMMIT;
      END TRY
    BEGIN CATCH
      ROLLBACK;
      END CATCH ;
 END
 go;
```

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

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.

Les verrous

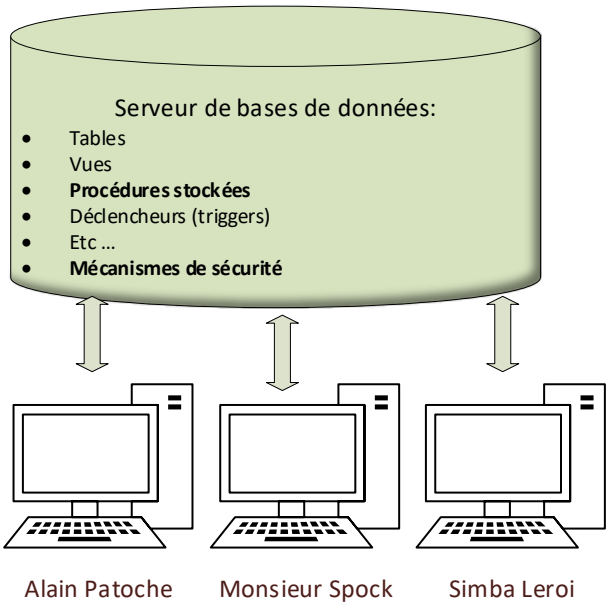
Le verrouillage est un mécanisme utilisé par le Moteur de base de données SQL Server pour synchroniser l'accès simultané de plusieurs utilisateurs à la même donnée.

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

Chapitre 7, 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.
  Sans procédure stockées : On voit toutes les tables, et les colonnes. Si la requête est longue alors c'est vraiment un problème.

```
SqlCommand objCommand = new SqlCommand("select nom, prenom,salaire, nomequipe from joueurs " +
    "inner join equipes on joueurs.CODEEQUIPE = equipes.CODEEQUIPE ", nomConnection);
```

  Avec procédure stockée : Le code est plus propre. Le nom des tables est « caché ».

```
SqlCommand objCommand1 = new SqlCommand("afficherJoueurs", nomConnection);
```

- Faciliter le débogage. Si la procédure stockée est compilée et s'est exécutée sur le serveur, alors si votre code C# (ou PHP) ne vous donne pas le résultat attendu, c'est qu'il n'est pas bon.
- Réutilisation de la procédure stockée.
- Possibilité d'exécuter un ensemble de requêtes SQL .
- Prévention d'injections SQL. Dans le code C# ou PHP ou autre, tout ce que nous voyons est le nom de la procédure (et ses paramètres). Les objets que la procédure manipule ne sont pas visibles pour les autres.
- Modularité. Facilite le travail d'équipe. Monsieur Spock et Alain Patoche peuvent travailler sur la même BD en utilisant les procédures stockées. Exemple Alain Patoche pourraient écrire le code C# pendant que Monsieur Spock écrit la procédure stockée qui permet de chercher une question aléatoirement à partir de la table Questions de la base de données.

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 1:

```
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. Cependant, 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 ='Alain',
@psal=22.5,
@pcodep ='sim',
@pnom ='Patoche';
```

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 : Exemple plus complet (réponse à la question 3-4 du laboratoire 1)

La procédure suivante, permet de créer une commande pour un client. Elle reçoit comme paramètres : Le IdClient, la date de la commande, le idArticle (l'article que le client veut acheter) ainsi que la quantité que l'on veut acheter.

Cette procédure fait les opérations :Lorsque la quantité commandée est plus petite à la quantité en stock alors :

1. Insérer dans la table commandes
2. Insère dans la table ligne_commande
3. Met à jour le montant de l'achat pour l'article
4. Met à jour la quantité en stock après achat.

Lorsque la quantité commandée est plus petite à la quantité en stock alors :On fait un ROLLBACK

Nous avons besoin de déclarer les variables suivantes :

@idcommande pour récupérer le idcommande et l'insérer dans la table Ligne_commande.

@prix, pour aller chercher le prix de l'artile pour mettre à jour le montant dans la table Ligne_commande

@quantite pour chercher la quantité en stock de l'article acheté pour la comparer à la quantité commandée (paramètre de la procédure) et mettre à jour la quantité en stock après achat.

```
CREATE PROCEDURE ajouterCommande(@idclient int,@date date, @idarticle int, @quaCde
int) as
BEGIN
declare
@idcommande int,
@prix money,
@quantite int;

select @prix = prix from Articles where idArticle=@idarticle;
select @quantite = quantite_stock from Articles where idArticle =@idarticle;
       begin transaction
       INSERT INTO Commandes(date_commande,idClient) values(@date,@idclient);
       select  @idcommande=@@IDENTITY;
       insert into Ligne_commande (idcommande,idArticle, quantiteCommande)
       values (@idcommande,@idarticle,@quaCde);

       if (@quantite <= @quaCde) rollback;
       else
              Begin
              update Articles set quantite_stock =quantite_stock-@quaCde
               where idArticle =@idarticle;
              update Ligne_commande set montant =@quaCde* @prix
               where idArticle =@idarticle;
              commit;
              end;
END;
go;
```

Exemple 5 : 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;
```

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;
```

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 (Exécution), nous avons besoin de préciser le schéma 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 compteretudiants() RETURNS INT
AS
BEGIN
DECLARE @total INT;
 SELECT  @total = COUNT(*) FROM  Etudiants ;
 RETURN @total;
END;
```

```
SELECT  dbo.compteretudiants();
---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 ChercherEMPLOYE (@code CHAR(3)) RETURNS TABLE
 AS
     RETURN(
          SELECT nom,prenom,salaire
          FROM employes
           WHERE @code  =CodeDep
          ) ;
```

Attention : 

1. In' y a pas de BEGIN et END pour une fonction qui retourne une table.

L'appel (Exécution) d'une fonction qui retourne une table est diffèrent. 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.

```
SELECT * FROM ChercherEMPLOYE('inf');
```

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

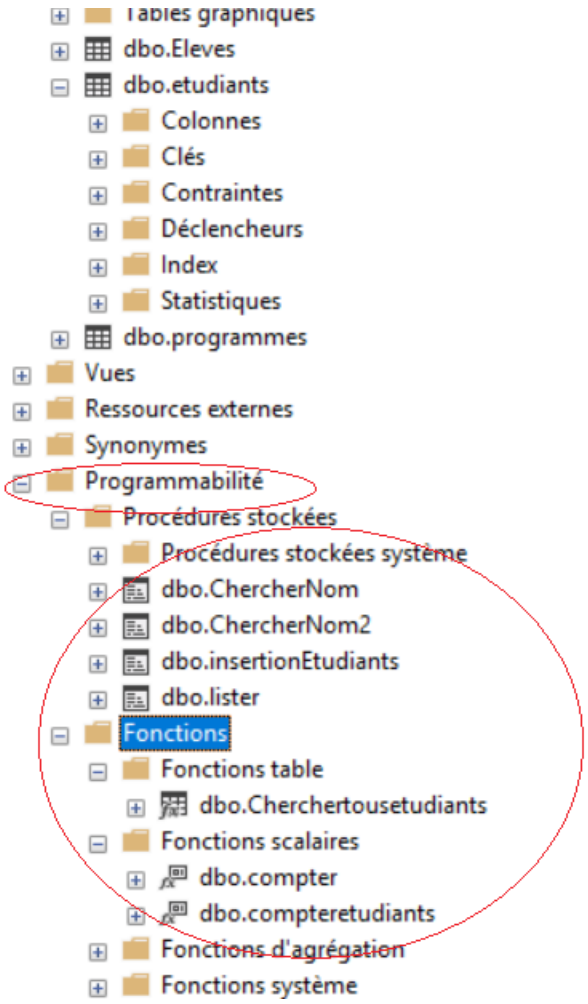
```
drop procedure ChercherNom2;
```

```
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
```

Autres exemples :

Procédure : *chercherQuestion*

Écrire une procédure stockée qui permet de ramener une question aléatoire d'une catégorie et qui met à jour le flag

```
create procedure chercherQuestion(@nomCategorie varchar(30)) as
begin
--- on déclare une variable pour récupérer le numéro de la question pigée
declare @numquestion smallint;
       select @numquestion = (select top 1 idQuestion
                              from QuestionTrivia Q inner join CategorieTrivia C
                              on Q.idCategorie=c.idCategorie
                              where nomCategorie = @nomCategorie
                              and flag='n' order by NEWID());

       select enonce from QuestionTrivia where idQuestion =@numquestion
       update QuestionTrivia  set flag ='o' where idQuestion = @numquestion

end;
```

Exécution :

```
execute chercherQuestion @nomCategorie ='sciences';
```

Procédure *: updtaeFlag*:

Met le flag à 'n' si toutes les questions d'une catégorie sont pigée

```
create procedure updateFlag(@nomCategorie varchar(30)) as
begin
declare @total int
select @total = count(*) from QuestionTrivia  Q
                              inner join CategorieTrivia C
                              on Q.idCategorie=C.idCategorie
                              where flag ='n' and nomCategorie =@nomCategorie;

 if (@total =0) update QuestionTrivia set flag='n' where
                                        idCategorie IN (
                                        select idCategorie from CategorieTrivia
                                        where nomCategorie =@nomCategorie)
end;
```

Exécution

```
execute updateFlag @nomCategorie ='sciences';
```

Procédure : *InserQuestion* :

Ajout d'une question et ses réponses

```
create procedure insertQuestion (@enonce varchar(100),
                                 @idcategorie char(1),
                                 @difficulte char(1),
                                 @repA varchar(100),
                                 @estbonneA char(1),
                                 @repB varchar(100),
                                 @estbonneB char(1),
                                 @repC varchar(100),
                                 @estbonneC char(1),
                                 @repD varchar(100),
                                 @estbonneD char(1)) as
Begin
declare @idQuetion int;
begin try
                    begin transaction
                    insert into QuestionTrivia(enonce,difficulte,idCategorie)
                    values (@enonce,@difficulte,@idcategorie);

                    select @idQuetion =@@IDENTITY;

                    insert into ReponseTrivia(laReponse,estBonne,idQuestion)
                    values(@repA, @estbonneA,@idQuetion);

                    insert into ReponseTrivia(laReponse,estBonne,idQuestion)
                    values(@repB, @estbonneB,@idQuetion);

                    insert into ReponseTrivia(laReponse,estBonne,idQuestion)
                    values(@repC, @estbonneC,@idQuetion);

                    insert into ReponseTrivia(laReponse,estBonne,idQuestion)
                    values(@repD, @estbonneD,@idQuetion);
                    commit;
end try
begin catch
      rollback;
      end catch
end;
```

Exécution

```
execute insertQuestion
@enonce ='Qui a écrit les misérables',
@idcategorie ='a',
@difficulte ='f',
@repA ='John Steimbeck',
@estbonneA='n',
@repB='Victor Hugo',
@estbonneB='o',
@repC ='Alain Patoche',
@estbonneC='n',
@repD ='Gustave Flaubert',
@estbonneD='n';
```

Procédure *deleteQuestion* :

Supprime une question et toutes ses réponses.

```
create  procedure deleteQuestion(@idQuestion smallint) as
begin
       begin transaction
       delete from ReponseTrivia where idQuestion =@idQuestion
       delete from QuestionTrivia where idQuestion =@idQuestion
       commit;
end;
```

Exécution :

```
execute deleteQuestion
@idQuestion=118;
```

Procédure avec un paramètre en OUT

Procedure *bonneReponse*,

Obtient la bonne réponse d'une question selon l'énoncé

```
create procedure bonneReponse(@enonce varchar(100),
                                                    @lareponse varchar(30) OUT) AS
begin
       select  @lareponse= lareponse
       from ReponseTrivia R
       inner join QuestionTrivia Q
       on R.idQuestion = Q.idQuestion
       where enonce like '%' + @enonce + '%'
    and estBonne ='o';
end;
```

Exécution :

```
declare
@enonce1 varchar(100) = 'Quelle est la couleur de Shrek',
@lareponse1 varchar(100) ;
execute bonneReponse
@enonce1,
@lareponse1 out;
print @lareponse1;
---
```

Chapitre 8, 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 DeleteReponse on questionTrivia instead of delete as
begin
declare @idQuestion int;
            select @idQuestion= idQuestion from deleted;
            delete from ReponseTrivia where idQuestion=@idQuestion;
            delete from QuestionTrivia where idQuestion =@idQuestion;
end;
```

Pour tester:

```
delete from QuestionTrivia where idQuestion =116;
```

Dans l'exemple précédent, les réponses aux questions doivent-être supprimées si la question est supprimée.
On demande au trigger de :
Au lieu de supprimer la question demandée , supprime d'abord les réponses correspondantes à la question, puis supprime la question.

Exemple 2, au lieu de faire une suppression en cascade, on fait une mise à jour.

```
create trigger updateQuestion on categorieTrivia instead of delete as
begin
  declare @iCategorie char(1);
              select @iCategorie= idCategorie from deleted;

              update  QuestionTrivia  set idCategorie= null
               where idCategorie =@iCategorie;

              delete from  CategorieTrivia where idCategorie =@iCategorie
end;
```

Dans l'exemple précédent, nous demandons au trigger de mettre à NULL le idCategorie dans la table QuestionTrivia si la catégorie est supprimée

Pour tester: (assurer d'abord que la colonne idCategorie accepte les valeurs NULL

```
alter table questionTrivia alter column idCategorie char(1) null;
delete from CategorieTrivia where idCategorie ='g';
```

Exemple 3

```
create TRIGGER ctrlSalairePermanent on EmpPermanent after  update
as

BEGIN
Declare
@empno int,
@ancienne money,
@nouvelle money;

      select @ancienne = Salaire from deleted where empno =@empno;
      select @nouvelle = Salaire from inserted where empno =@empno;
      IF (@ancienne  > @nouvelle)
            begin
            rollback;
            RAISERROR (15600,-1,-1, 'ne pas diminuer les salaires');
            end;
END;
```

Exemple 4

Ce trigger CTRLReponses vérifie si le nombre de bonnes réponses est >1 ou que les questions ont plus de 4 réponses si tel est le cas, il annule la transaction

```
create  trigger CTRLReponses on ReponseTrivia after insert, update as
begin
declare @idQuestion int,
            @nbReponse smallint,
            @totalBonneRep smallint;

      select @idQuestion = idQuestion from inserted;
      select @nbReponse = count(*) from ReponseTrivia
                                  where idQuestion =@idQuestion;

      select @totalBonneRep = count(idQuestion) from ReponseTrivia
                                                where idQuestion =@idQuestion
                                                 and estBonne ='o';

      if (@totalBonneRep <> 1 or @nbReponse > 4) rollback;

end;
```

Pour tester

1- Pour une question, on insère 4 réponses mais deux bonnes réponses;
2- Après on fait le test : pour une question, on insère 5 réponses avec une seule bonne réponse
3- Enfin, pour une question, on insère 4 réponses avec une seule bonne réponse

```
begin transaction
 insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Chat ','o',121);
 insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Chien','o',121);
 insert into ReponseTrivia(laReponse,estBonne,idQuestion)values('Cheval','n',121);
 insert into ReponseTrivia(laReponse,estBonne,idQuestion)values('Chacal','n',121);
 --insert intoReponseTrivia(laReponse,estBonne,idQuestion)values('Chèvre','n',121);
commit;
```

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 ramené 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 EmpclgDB;
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 Gravité,
   ERROR_STATE() as etat,@@TRANCOUNT

end catch;
```

	message	Gravité	etat
1	An invalid parameter or option was specified for procedure 'Le salaire ne doit pas être révisé à la baisse'.	15	1

Message : représente le message défini par l'utilisateur. Au maximum 2047 caractères.

Vous pouvez également laisser le soin au SGBDR d'utiliser ses propres paramètres.

Le trigger de l'exemple 4, pourrait s'écrire de la manière suivante pour renseigner un peu plus l'utilisateur. On indique deux messages erreurs. Un message erreur lorsque le nombre de bonnes réponses est plus grand que 1, et un autre lorsque le nombre de réponses d'une question est plus grand que 4.

```
create trigger CTRLReponses on ReponseTrivia after insert, update as
begin
declare @idQuestion int,
              @nbReponse smallint,
              @totalBonneRep smallint;

       select @idQuestion = idQuestion from inserted;
       select @nbReponse = count(*) from ReponseTrivia
                                    where idQuestion =@idQuestion

       select @totalBonneRep = count(idQuestion) from ReponseTrivia
                                                  where idQuestion =@idQuestion
                                                 and estBonne ='o';
       if (@nbReponse> 4)
              begin
                     rollback;
                     RAISERROR (15600,-1,-1, 'maximum 4 réponses par question.');
              end;
       else if (@totalBonneRep > 1)
              begin
              rollback;
              RAISERROR (15600,-1,-1, 'Limite de 1 bonne réponse par question');
              end;

end;
```

Et on pourra tester de la manière suivante

```
begin try
 begin transaction
 insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Chat','o',122);
 insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Chien','n',122);
 insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Cheval','n',122);
 insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Chacal','n',122);
--insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Chevre','n',122);
 commit;
 end try
 begin catch
              select ERROR_MESSAGE() as message, ERROR_SEVERITY() as Gravité,
              ERROR_STATE() as etat,@@TRANCOUNT as nb_transaction

end catch;
```

Attention : 

Les triggers sont définis sur une table , ce sont donc des objets de la table, tout comme une colonne, une contrainte…

Activer /désactiver un trigger

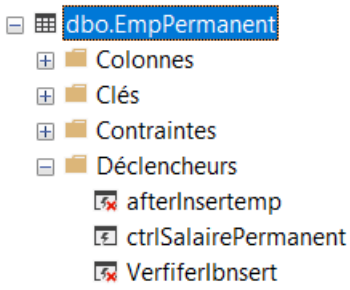
Utiliser la commande DISABLE pour désactiver temporairement un trigger

```
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 désactivé va toujours exister dans le système mais ne fait rien. (sans action). Dans Management Studio, il est marqué en rouge.



Pour déscativer TOUS les triggers sur une table :

```
disable   trigger all on  [dbo].[EmpPermanent];
```

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, 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 pk_livre 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 fk_Livre 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--
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', 'Algébre Linéarie','Français',2013,400);


---insertion dans Chapitres
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');


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

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 [EmpclgDB]
GO
/****** Object:  Trigger [dbo].[deletecascdeDepartement] ****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
ALTER  trigger [dbo].[deletecascdeDeparetement] 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 (points clés):

1. Pour garantir l'intégrité de données en général et référentielle en particulier, faîtes-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 OR ALTER PROCEDURE supprimerDepartement(@deptno INT)
AS
BEGIN
BEGIN TRY
      BEGIN TRANSACTION
--Enlever les nos de département pointant vers celui que l'on veut supprimer
      UPDATE EmpPermanent SET deptno=NULL WHERE deptno=@deptno;
--Enlever le département de la table Departements
      DELETE FROM Departements WHERE deptno=@deptno;
      COMMIT;
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 faîtes attention

Chapitre 9, les Curseurs

Définition :

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 nom, prenom FROM Clients;
```

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

1- Ouvrir le curseur avec **OPEN**.
2- Lire le curseur avec **FETCH ……INTO** 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.

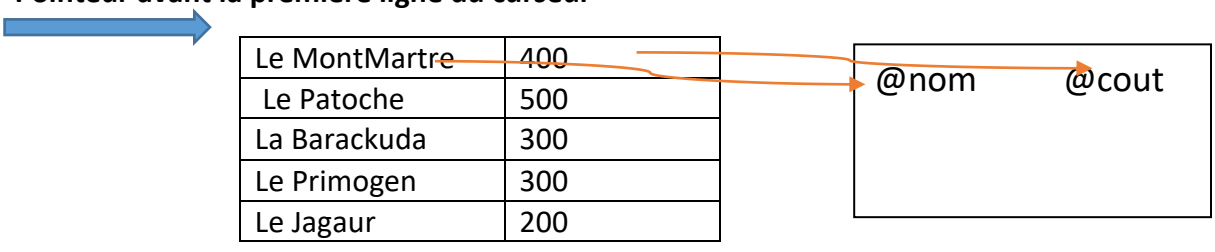
Exemple1: Affichage uniquement

```
DECLARE @nom varchar(30), @cout int;
DECLARE cur_circuit CURSOR FOR SELECT nomcircuit, coutcircuit
FROM circuits;
BEGIN
  OPEN cur_circuit ;
  PRINT CONCAT('nom circuit','---','cout du circuit');
-- on initialise les variable @nom et @cout avec le premier
FETCH(la première ligne)
  FETCH NEXT FROM cur_circuit INTO @nom, @cout;
-- Tant que le FETCH se fait normalement
  WHILE @@FETCH_STATUS = 0
      BEGIN
      PRINT CONCAT(@nom,' -------'  ,@cout);
      FETCH NEXT FROM cur_circuit INTO @nom, @cout;
  END;
CLOSE cur_circuit;
DEALLOCATE cur_circuit;
END;
```

Explications:

1- Lorsque le curseur `cur_circuit` est déclaré avec FOR SELECT
   a. il contient le résultat de la requête SELECT.
   b. Il y a un pointeur avant la première ligne du curseur.
2- On ouvre le curseur avec la commande OPEN.
3- La commande `FETCH NEXT FROM cur_circuit INTO @nom, @cout` fait avancer le pointeur à la prochaine ligne. Après le `FETCH NEXT` la flèche bleue avance à la première ligne du curseur
4- La première ligne du curseur contient des données. Ces données sont affectées respectivement à @nom, @cout.

Pointeur avant la première ligne du curseur



5- Tant que le curseur n'est pas vide (`WHILE @@FETCH_STATUS = 0`) on passe à la ligne suivante avec `FETCH NEXT FROM cur_circuit INTO @nom, @cout`
6- On ferme le curseur
7- On supprime la référence au curseur.

Affichage résultat

```
nom circuit---    coût du circuit
Le MontMartre -------------400
Le Patoche ----------------500
Le Barackuda --------------300
Le Primogene --------------300
Le Jaguar -----------------200
```

Attention: 

Il est inutile d'utilser un curseur pour ne faire qu'afficher son contenu. La sortie de votre SELECT est suffisante. Il est même déconseillé d'utilser un curseur que pour l'affichage du résultat puisque la vitesse d'affichage sera grandement ralentie.

Exemple 2 : Curseur pour UPDATE

```
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<300 UPDATE circuits set coutcircuit = coutcircuit+
         (coutcircuit*0.1) WHERE CURRENT OF Cout_cursor ;

         ELSE IF @cout BETWEEN 300 AND 450 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;
```

Explications : Le même principe qu'avec l'exemple plus haut. Mais au lieu d'afficher le contenu du curseur on met à jour la base de données selon le contenu du curseur.

Exemple 3, curseur SCROLLABLE

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 , nbEtoile FROM Monuments
ORDER BY nbEtoile 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 numéro 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('la ligne immédiatement avant la position courante');
  FETCH PRIOR FROM Curmonument into @nom,@nb;
  PRINT CONCAT(@nom,'----', @nb)

  PRINT('la ligne 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
```

Exemple 4, Laboratoire 1, question 9

```
DECLARE @prix int;
DECLARE @idArticle int;
DECLARE prix_cursor CURSOR FOR SELECT idArticle,prix FROM Articles;
BEGIN
      OPEN prix_cursor ;
      FETCH NEXT FROM prix_cursor INTO @idArticle,@prix
      WHILE @@FETCH_STATUS = 0
            BEGIN
            UPDATE Ligne_commande set Montant  =quantiteCommande*@prix
                  where idArticle =@idArticle;

            FETCH NEXT FROM prix_cursor INTO @idArticle,@prix
         END;
CLOSE prix_cursor;
DEALLOCATE prix_cursor;
END;
```

Exemple 5, Le panier d'achats

Voici le contenu de la table Articles de votre BD StockClg;

	idArticle	descriptions	prix	quantite_stock
1	1	Imprimantes Laser HP 8000	700,00	25
2	2	Écrans tactiles 19p	550,00	55
3	3	Routeurs sans fils Azus	200,00	90
4	4	Disques durs SSD 500 Go	175,00	45
5	5	Chaises de bureau	400,00	20
6	6	Cet Article	400,00	3
7	7	Mon article	100,00	4

Voici le contenu du panier d'achat des clients.

Résultats Messages

	idArticle	idClient	qteAchat
1	1	1	5
2	2	1	10
3	3	1	15
4	4	2	10
5	1	2	15

Ce que nous voulons est la chose suivante.

Lorsque le panier d'achat d'un client est supprimé, alors il faudra **remettre** les quantités dans la table Articles.

Si le panier du client 1 (idClient =1) est supprimé alors :

1. L'article numéro 1 sera augmenté de 5 dans la table Articles
2. L'article numéro 2 sera augmenté de 10 dans la table Articles
3. L'article numéro 3 sera augmenté de 15 dans la table Articles

Pour faire cette opération, nous allons écrire une procédure stockée qui va utiliser un curseur.

```
create procedure supprimerPanier(@idClient smallint) as
begin
declare
             @qtAchat int,
             @idArticle int;

declare      curPanier cursor for select idArticle, qteAchat from PanierAchat
              where idClient  =@idClient;
----avant de supprimer , on remet les quantités du panier dans la table Articles.

--- on ouvre le curseur
       open curPanier;
       -- on avance le pointeur au premier enregistrement. On lit, et
       --on met le contenu du curseur dans deux variables @idArticle,@qtAchat;
       fetch next from curPanier into @idArticle,@qtAchat;
       while @@FETCH_STATUS=0
              begin
on met à jour la table Articles en augmentant les quantités en stocks par les ;
quantités du panier pour chaque article du panier ;
              update Articles  set quantite_stock = quantite_stock + @qtAchat
                     where idArticle =@idArticle;
              fetch next from curPanier into @idArticle,@qtAchat;
              end;
on supprime le panier
       delete from PanierAchat where idClient =@idClient;
on ferme le curseur
       close curPanier;
on libère les ressources
       DEALLOCATE curPanier;
end;
```

Chapitre 10, 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 une base de données.

Par défaut, TOUS les SGBD entretiennent un index primaire qui est l'index crée 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 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 indexes sont des accélérateurs, trop d'index ralenti 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 mis à jour.

Éviter les indexes sur les colonnes :

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

Types d'index :

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

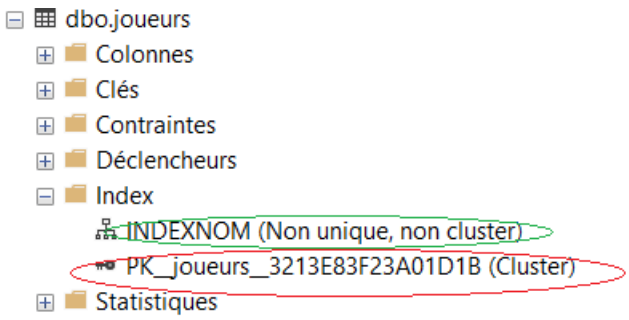
Les CLUSTERED INDEX :

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 CLUSTERD 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

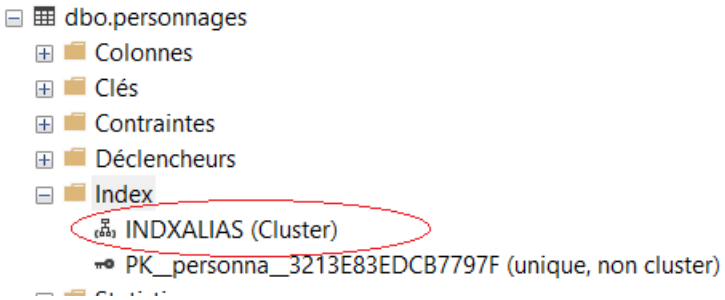


Si vous voulez mettre un autre index Cluster sur votre table il faudra :

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

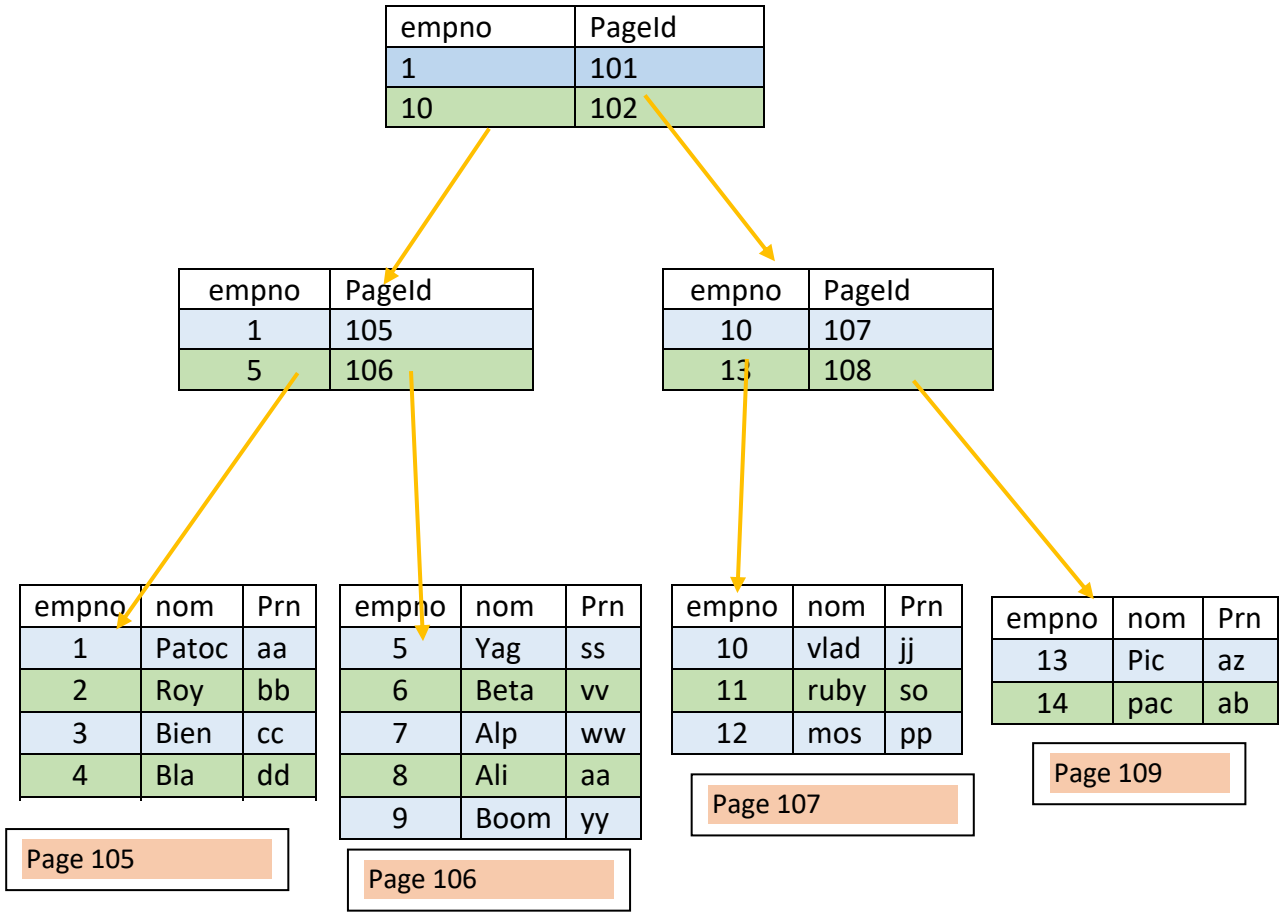
```
create table personnages
(
idPerosonnage int identity(1,1),
alias varchar(10)  NOT NULL unique,
nom varchar(30) not null,
descriptions varchar(60) not null,
constraint pk_personnage primary key nonclustered (idPerosonnage)
);
```

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



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 indexsurType 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_constrainte*
UNIQUE (*nom_colonne*)

ALTER TABLE *nom_table* ADD CONSTRAINT *nom_constrainte*
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 11, 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 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`BULK INSERT`.
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

Rôles	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.
Public	Permet de faire un USE sur la BD

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

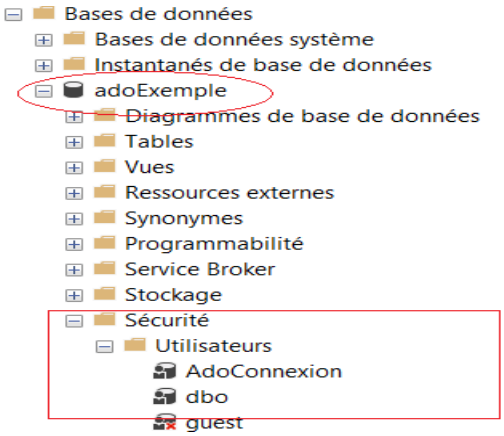
Par l'interface SQL Server Management Studio :

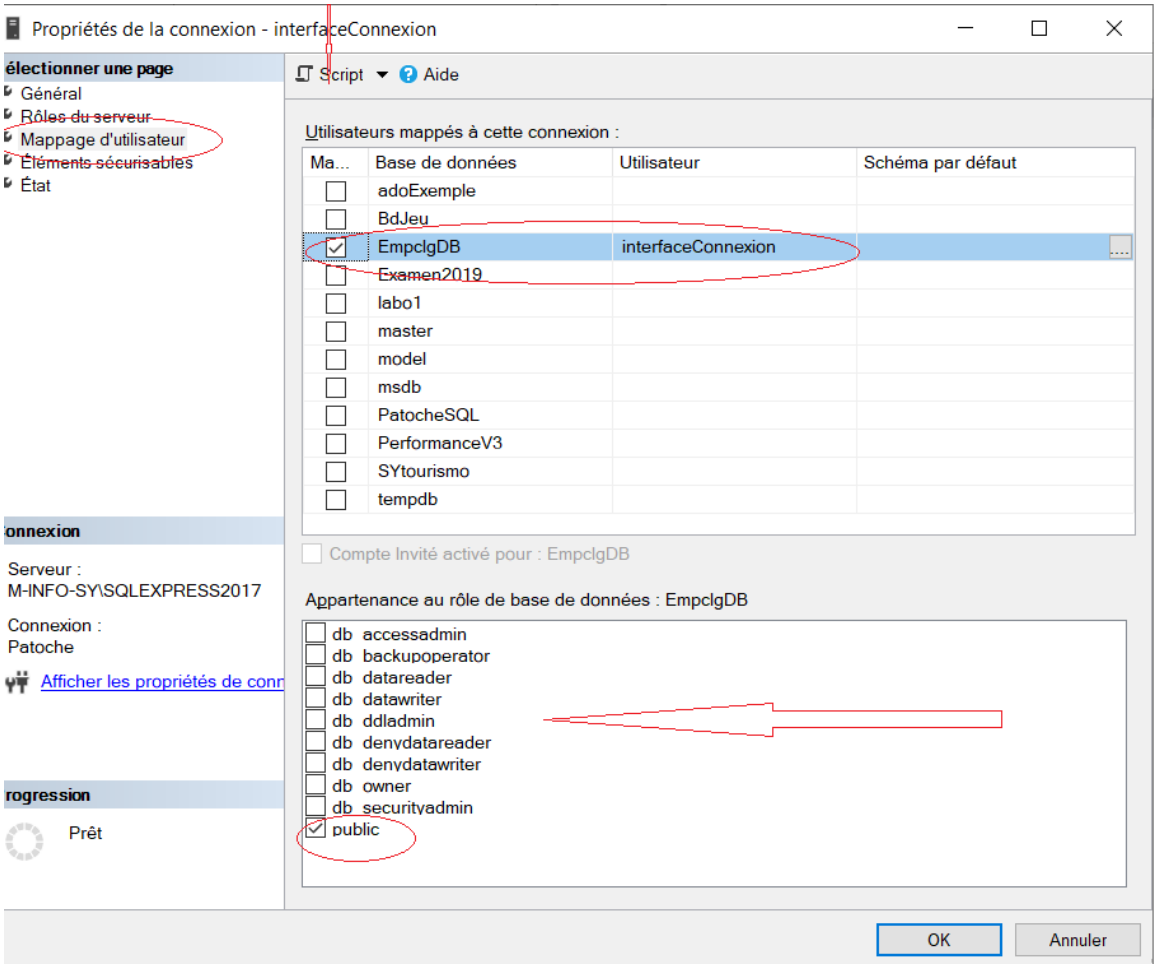
On suit les mêmes étapes pour créer une connexion, puis avant de cliquer sur OK, il faudra mapper un utilisateur à cette connexion

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.



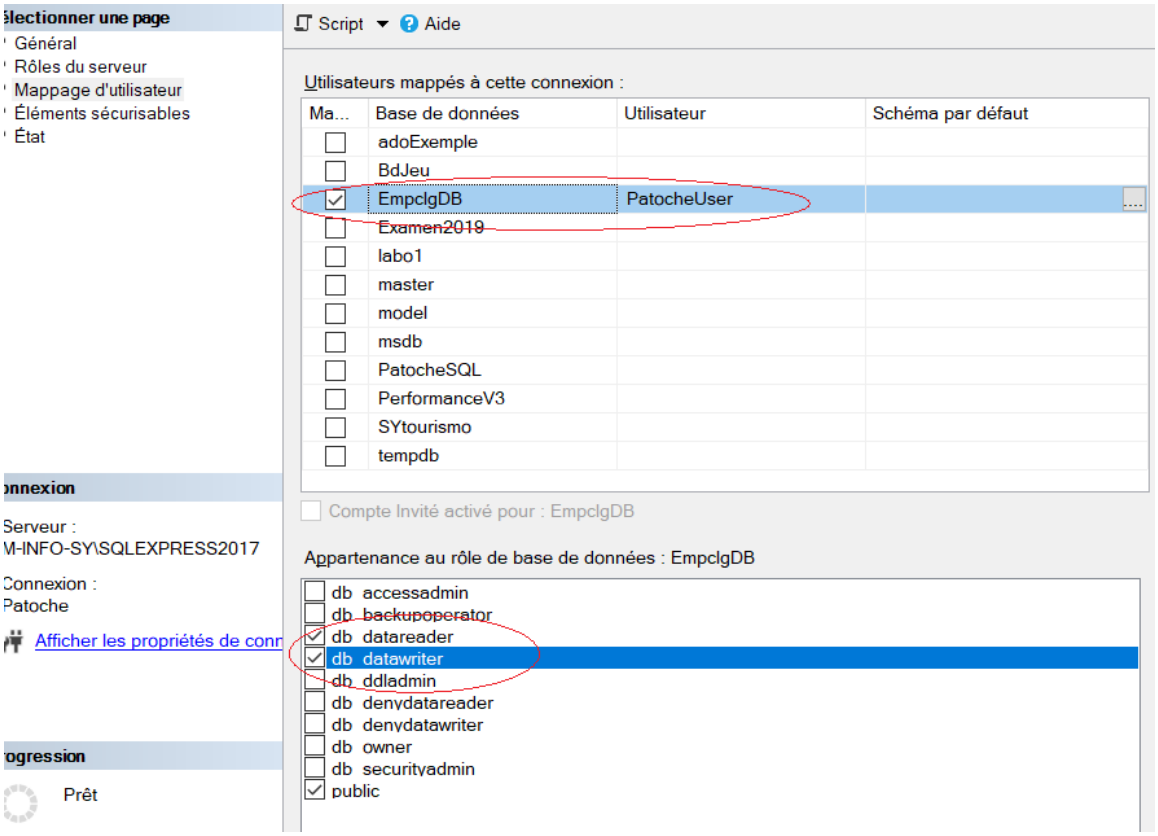


Dans cette figure on voit que :

- 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
```



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'usager AdoConnexion a le droit de faire select, update, insert et Delete sur toutes les table 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'usager 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 ROLES 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 role sysadmin**)

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

create user PatocheUser for login logPatoche;
```

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

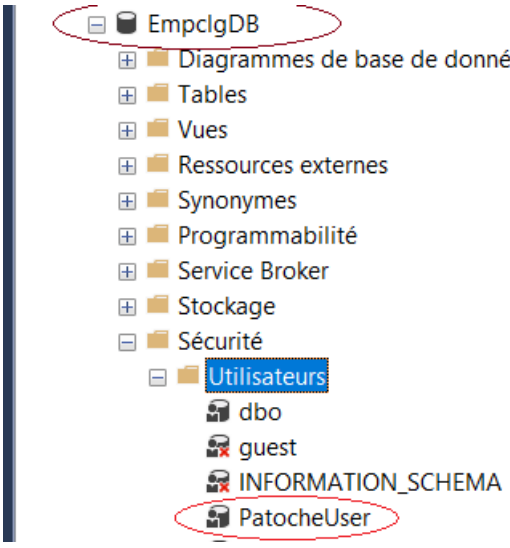
Pour créer un utilisateur non orphelin, donc rattachée à une base de données EmpclgDB par exemple, il faudra :

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

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

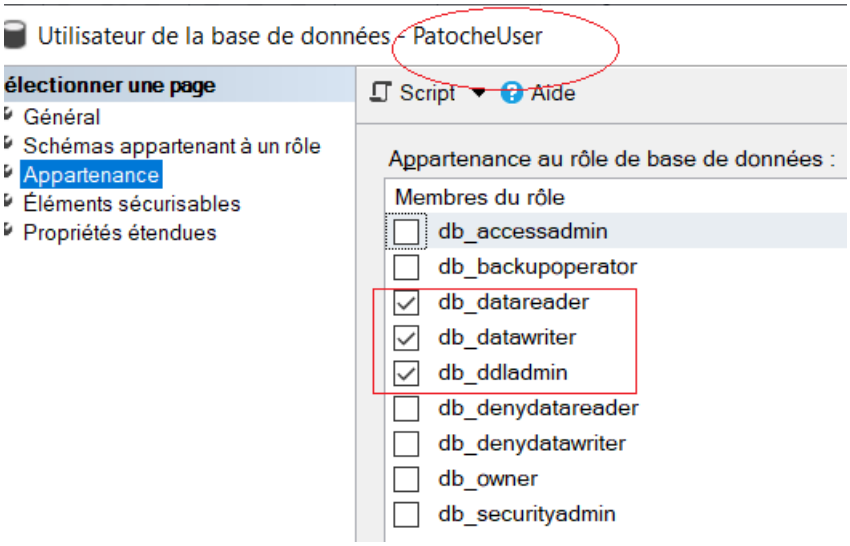
Exemple :

```
use EmpclgDB;
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_personnne 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'executer n'importe quelle procédure.

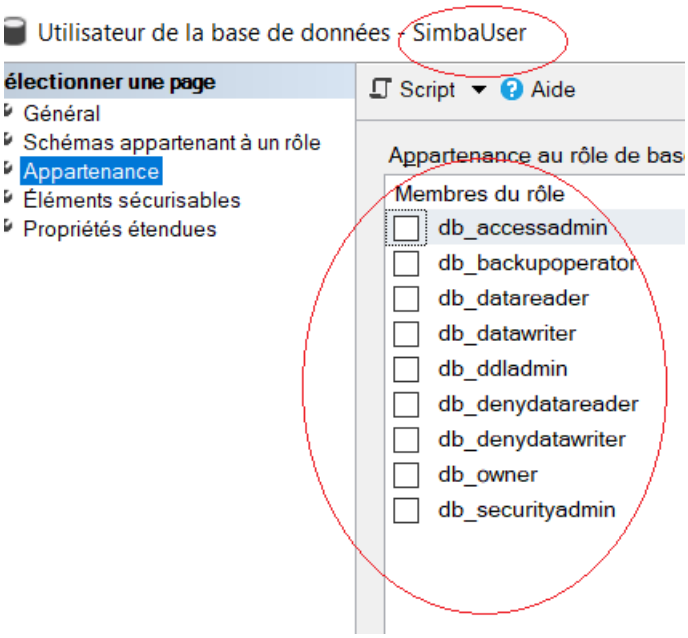
Les commandes GRANT, REVOKE et DENY

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 peu donner le même privilège à un autre utilisateur.

Les roles creés par les utilisateurs. (pas ceux prédéfinis).

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

Situation : vous êtes une équipe de 10, donc **10 users** à travailler sur le même projet. Vous utilisez donc des tables de ce projets. 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 role roleprojet avec un certain nombre de droits.

Tous ce que nous avons à faire c'est d'affecter le role aux usager qu'on veut.

```
EXEC sp_addrolemember roleprojet, RubyUser;

EXEC sp_addrolemember roleprojet, RemiUser;
```

Maintenant… imaginez que vous avez oublié de donner le privillège 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 faires certaine 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 chaines 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 illisible ou impossible 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 à 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 (chaine de caractères de longueur variable) et qui génère en sortie une chaine 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 chaines 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;

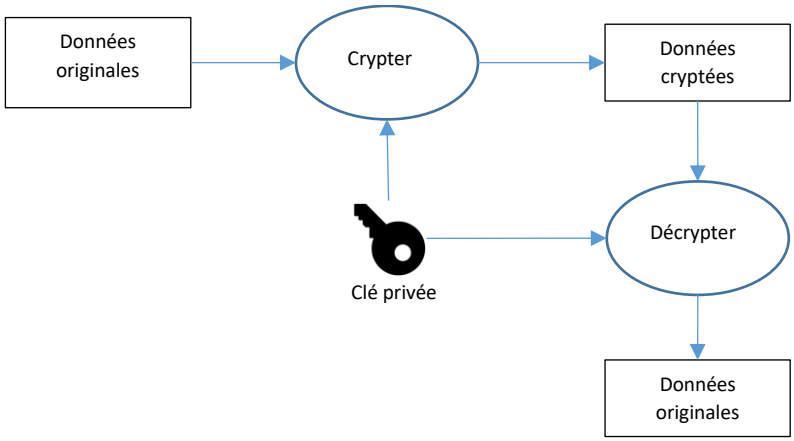


Figure 1: Chiffrement symétrique

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 détendeur 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);

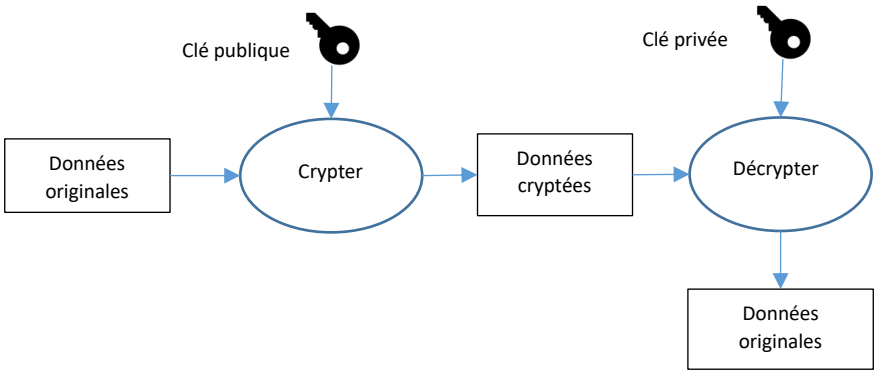


Figure 2: Chiffrement asymétrique

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 pas 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 3: 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 4: 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 chaine 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 crédite '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 5: exemple de chiffrement symétrique avec **ENCRYPTBYPASSPHRASE.***

Fonction de hachage

HASHBYTES

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

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

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

Figure 6: 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 ENCRYPTBYPASSPHRASE

```
alter table fournisseurs add carteCryptee varbinary(128);

create procedure ajouterCarteCredit (@id int, @motdepasse varchar(30),
@carteNonCrypte varchar(20)) as
begin
declare @motdepasseHash varbinary(128);
       select @motdepasseHash = mpasse from fournisseurs where idfournisseur =@id;
       if(HASHBYTES('SHA2_512',@motdepasse ) = @motdepasseHash)
       begin
              update fournisseurs set carteCryptee  =
              ENCRYPTBYPASSPHRASE(@motdepasse, @carteNonCrypte , 0)
               where  idfournisseur =@id;
       end;
end;
```

Pour décrypter : (vous pouvez utiliser une procédure ou une fonction)

```
SELECT nom, adresse, carteCryptee AS 'carte encrypte',
CONVERT(varchar, DECRYPTBYPASSPHRASE('passww$33',carteCryptee))
AS 'carte decryptée' FROM fournisseurs where idfournisseur =2;
```

Autre exemple chiffrement par clé symétrique sans certificat

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

```
 alter table fournisseurs add nassCrypteByKey varbinary(128);

---- Encryption par clé symetrique
CREATE SYMMETRIC KEY SymmetricKeyNass
WITH ALGORITHM = AES_128
ENCRYPTION BY password ='local$33';
GO

OPEN SYMMETRIC KEY SymmetricKeyNass
Decryption BY password ='local$33';
GO

UPDATE fournisseurs SET nassCrypteByKey = EncryptByKey
(Key_GUID('SymmetricKeyNass'),'999-120-506') where idfournisseur=1;

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

--on affiche pour voir que les données du Nas ont été cryptée.

```
select * from fournisseurs;
```

Messages

nom	adresse	mpasse	carteCryptee	nassCrypteByKey
le iga	laval etc	0x2E33B2AB38861BF326645AC58A85B5BDCA8AFDE897D7F5A...	0x02000000A42D3DBBE645E09C7BB656897A1DC71D6A48...	0x0015A6A25D1E604DBD95530AE54D439102000000CBF120...
le metro	rue de la liune ici	0x97780E3B6AED0C1764151E77B2D0ED539675F1CFF52FE64C...	0x020000008FA8DD454CEAF7690BC13059915B02921CF49...	NULL
Lafleur	pas loin	0xF8AE481DCC4A9D8AC4317BAC2FCBA7729B2CF0DABB5EF4...	NULL	NULL
un fournisseur	blaba	0x2E33B2AB38861BF326645AC58A85B5BDCA8AFDE897D7F5A...	NULL	NULL

-on utilise à nouveau la clé pour décrypter

```
OPEN SYMMETRIC KEY SymmetricKeyNass
Decryption BY password ='local$33';
GO

-- on affiche le nass décrypté
SELECT nom, nassCrypteByKey AS 'Nas encrypté',
CONVERT(varchar, DecryptByKey(nassCrypteByKey)) AS 'nas decrypté'
FROM fournisseurs where idfournisseur=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

https://learn.microsoft.com/fr-fr/sql/relational-databases/security/authentication-access/database-level-roles?view=sql-server-ver16