Support de Cours

REQUÊTES TRANSACT-SQL DANS MICROSOFT SQL SERVER 2000

Stéphane N'GUESSAN
Groupe Pigier – Abidjan
Version 1.1 du 29/04/05

Table des matières

CHAPITRE 0 : INTRODUCTION

Bienvenue dans ce cours de « Requêtes Transact-SQL dans SQL Server 2000 ».

I) Objectifs

Ce cours donne tous les éléments utiles pour créer une base de données dans SQL Server 2000 en utilisant uniquement les instructions du langage Transact-SQL.

II) Certification Microsoft

Ce cours est un pré requis nécessaire, mais pas suffisant pour passer les examens 70-228 (Administration de SQL Server 2000) et 70-229 (Implémentation de SQL Server 2000) de Microsoft.

III) Conventions

Dans ce document, nous utiliserons les conventions suivantes :

- lorsque nous écrivons « cliquez sur Fichier->Ouvrir » cela signifie « Déroulez le menu Fichier et sélectionnez l'élément Ouvrir » ;
- dans la syntaxe, les éléments entre crochets [ ] sont facultatifs ; les éléments entre accolades { } sont des options. Le séparateur d'options est le caractère |.

IV) Informations

Bonne Lecture !

Stéphane N'GUESSAN
snb@powernet.ci
MCP Visual Basic .NET, MCDBA SQL Server 2000, MCSE 2000
Développeur SQL Server depuis 1999
Groupe Pigier - Abidjan

CHAPITRE 1 : PRÉSENTATION DE TSQL

Le langage **TSQL** (**T**ransact **S**trutrured **Q**uery **L**anguage) permet de normaliser le développement d'applications liées aux **B**ases de **D**onnées. TSQL est une extension du langage SQL développé par IBM (International Business Machine) dans les années 1970. C'est un langage procédural par opposition à SQL qui est un langage déclaratif.

I) Historique

Le langage SQL est né à la suite des travaux du mathématicien *Codd*. Historiquement, les années suivantes sont importantes :

- 1970 : IBM développe SQL
- 1986 : Normalisation SQL/86
- 1989 : Normalisation SQL/89
- 1992 : Normalisation ANSI SQL/92
- 1996 : Microsoft adopte TSQL

II) Les différents types d'instructions

Le langage SQL possède trois types d'instructions :

- le **Langage de Définition des Données** (LDD ou DDL en anglais) utilisé pour la création, la modification et la suppression des objets de base de données (tables, vues, etc.). Il utilise principalement les instructions CREATE, ALTER et DROP ;

  Exemple: CREATE DATABASE Test
- le **Langage de Manipulation des Données** (LMD ou DML en anglais) qui permet de sélectionner, d'ajouter, de modifier et de supprimer des données dans les objets de base de données (tables, vues, etc.). Il utilise principalement les instructions SELECT, INSERT, UPDATE et DELETE ;

  Exemple: SELECT * FROM Produits
- le **Langage de Contrôle des Données** (LCD ou DCL en anglais) utilisé pour la gestion des protections d'accès aux données. Il utilise principalement les instructions GRANT, DENY et REVOKE.

  Exemple : GRANT Select ON Produits TO Steve

III) Les Directives

Les directives indiquent comment traiter les instructions TSQL. Les principales directives sont :

- **USE « base de données »** : permets de préciser sur quelle base de données vont porter les instructions qui suivent.

  Exemple : USE Master
- **GO** : SQL Server interprète l'instruction GO comme un signal pour exécuter les instructions TSQL qui précèdent. Une commande GO termine un lot d'instructions TSQL. Certaines instructions TSQL doivent s'exécuter comme des lots, dans ce cas, l'utilisation de la commande GO est obligatoire.

  Exemple :
  ```
  USE Northwind
  SELECT * FROM Product
  GO
  ```
- **PRINT** : est une instruction permettant de générer une ligne en sortie de procédure.

  Exemple : `PRINT @@VERSION`
- **EXEC[UTE]** : est utilisée pour exécuter une fonction ou une procédure stockée.

  Exemple : EXEC getdate()

IV) Éléments de syntaxe de TSQL

Le langage TSQL comporte des éléments que l'on retrouve dans la plupart des langages évolués :

- les types de données : ils déterminent la nature du contenu des objets tels que les colonnes, les variables, les paramètres, etc.

  Numérique : Ce type de données représente des valeurs entières ou décimales.

```
      'Entier
Int          4o    De -2^31 à 2^31-1
Tinyint      1o    De 0 à 255
Smallint     2o    De -32768 à 32767
Bigint       8o    De -2^63 à 2^63-1
      'Décimales
Decimal      supporte 38 digits
Numeric      idem à Decimal
      'Monnétaire
Money        8o    ±922337 Milliards
Smallmoney   4o    ±21747 Milliards
      'à virgule flottante
```

```
Float          8o    ±1.74E308
Real           4o    ±3.49E38
      'Booleen
Bit            1o    valeur binaire (0 ou 1)
      'Spécial
UniqueIdentifier  16o   Définit une colonne comme
identificateur unique globale (GUID)
```

Dates : Ce type de donnée représente une date ou une durée.

```
DateTime           8o De 01/01/1753 à 31/12/9999
SmallDateTime      4o De 01/01/1900 à 06/06/2079
```

Caractères : Ce type de données est utilisé pour représenter les chaînes de caractères de longueur fixe ou variable.

```
Char          taille fixe n'excédant pas 8Ko
Varchar       taille variable n'excédant pas 8Ko
Nchar         unicode taille fixe n'excédant pas 4Ko
Nvarchar      unicode taille var. n'excédant pas 4Ko
Text          taille variable pouvant excéder 8Ko
Ntext         unicode taille var. pouvant excéder 4K
```

Binaire : Ce type de données est très proche du type de données de caractères en termes de stockage et de structure, mais le contenu des données est traité sous forme d'une série de valeurs binaires.

```
Binary        taille fixe n'excédant pas 8Ko
VarBinary     taille variable n'excédant pas 8Ko
Image         taille variable pouvant excéder 8Ko
```

- les identificateurs : ce sont les noms des objets tels que les bases de données, les tables, les colonnes, etc.
    - identificateurs standard

    Les identificateurs standard peuvent contenir de 1 à 128 caractères, notamment des lettres, des symboles (_, @ ou #) et des nombres. La première lettre doit être alphabétique et aucun espace n'est autorisé dans le nom. Ils ne doivent correspondre à aucun mot-clé du langage TSQL.
    - identificateurs délimités

    Un identificateur qui ne respecte pas une ou plusieurs des règles de mise en forme des identificateurs standard doit toujours être délimité. Les délimiteurs sont [ ] et " ".

```
Exemple : SELECT * FROM "Liste des Etudiants"
```



```
DROP DATABASE [123ABC]
```

Le symbole @ commence le nom de toute variable locale et @@ désigne les variables globales. Le symbole **#** commence le nom de toute table temporaire et **##** désigne les tables temporaires globales.

```
SELECT *
FROM ##tb_tempo
```

- les opérateurs : permettent de créer des expressions complexes à partir d'expressions simples. Parmi les opérateurs, nous avons :
    - opérateurs arithmétiques : + , - , / , * , %
    - opérateurs logiques : NOT , AND , OR , XOR
    - opérateurs de comparaison : > , < , = , >= , <= , <>
- les variables : éléments permettant de stocker une valeur en vue de son utilisation future. Une variable locale est définie par l'utilisateur par l'instruction **DECLARE**, se voit affecter une valeur initiale dans une instruction **SET** ou SELECT est utilisée dans l'instruction ou le lot.

```
USE Northwind
DECLARE @EmpID int,@Name char(20)
SET @EmpId = 101
SELECT @Name = 'Dupont'
```

- les commentaires : ce sont des parties de texte insérées dans un jeu d'instruction pour en expliquer le but. Les commentaires ne sont pas exécutés.
    - Commentaire en ligne

```
SELECT productname, unitsinstock - unitsonorder
-- Calcule l'inventaire  supplierID
FROM products
GO
```

    - Commentaire en bloc

```
/* Ce programme extrait toutes les lignes de la
table products, et affiche le prix unitaire, le
prix unitaire augmenté de 10 % et le nom du
produit.*/
USE Northwind
SELECT unitprice, (unitprice * 1.1), productname
FROM Product
GO
```

- les scripts : c'est un ensemble d'instructions TSQL

CHAPITRE 2 : UTILISATION DES OUTILS DE REQUÊTES

SQL Server 2000 fournit plusieurs outils de requête que vous pouvez utiliser pour exécuter des scripts Transact-SQL.

I) Présentation de SQL Server

SQL Server 2000 ajoute de nombreuses fonctionnalités par rapport à ces versions antérieures (6.5 et 7.0), dont la prise en charge de XML (eXtended Markup Language), Datawarehouse (Entrepôt de données), OLAP et OLTP (traitement de données en ligne)

1. Composants client-serveur

L'architecture de SQL Server est divisée en :

- une **partie serveur** : gestion des bases de données, répartition des ressources entre les différentes demandes des clients (effectue le traitement lourd) ;
- une **partie cliente** : affichage des résultantes de requêtes (effectue le traitement léger (renvoi du traitement lourd)).

Les communications entre le serveur et le client peuvent s'effectuer sur différents protocoles réseaux dont TCP/IP, Canaux nommés, IPX/SPX.

Les fonctionnalités de SQL Server reposent sur l'exécution de plusieurs services :

- **MSSQLServer** gère les requêtes, les transactions et assure l'intégrité des données ;
- **MSSQLServerAgent** permet l'exécution des tâches planifiées.

2. Bases de données SQL Server

SQL Server héberge deux types de bases de données : les **bases de données systèmes** et les **bases de données utilisateurs**. Les bases de données système contiennent des informations nécessaires au bon fonctionnement de SQL Server. Les bases de données utilisateur sont les bases de données créées et utilisées par les utilisateurs.

Lors de l'installation de SQL Server, le programme d'installation crée plusieurs bases de données système :

- **master** : contrôle les bases de données utilisateurs et le fonctionnement global de SQL Server en effectuant le suivi d'informations telles que les comptes d'utilisateur, les

variables d'environnement configurables et les messages d'erreur du système ;
- **model** : utilisée pour la création de nouvelles bases de données utilisateur ;
- **tempdb** : offre une zone de stockage pour les tables temporaires et les autres besoins de stockage temporaire ;
- **msdb** : offre une zone de stockage pour les informations de programmation et l'historique des travaux.

Selon les options choisies, le programme d'installation peut créer les bases de données utilisateurs suivantes :

- **Pubs** : propose un exemple de base de données comme outil d'apprentissage ;
- **Northwind** : propose un exemple de base de données comme outil d'apprentissage avec un grand nombre de données.

3. Éditions de SQL Server

Il existe plusieurs éditions de SQL Server 2000 :

- **Entreprise** : en environnement de production Client/Serveur, cette version regroupe toutes les fonctionnalités avec la prise en charge de sites Web, OLTP, le DataWarehousing, etc ;
- **Entreprise d'évaluation** : Cette édition est identique à la précédente mais n'est utilisable que 120 jours. Elle est téléchargeable gratuitement ;
- **Standard** : cette édition regroupe toutes les fonctionnalités de l'édition Entreprise, sauf pour ce qui est des sites Web, OLTP et le DataWarehousing. Cette édition convient à de petites exploitations, avec des bases de données plus légères ;
- **Développeur** : cette édition, identique à l'édition entreprise, est réservée au développement d'applications basées sur SQL Server, aux tests et non à l'exploitation ;
- **Windows CE** : cette édition allégée permet le stockage de données sur des périphériques exploitant Windows CE. Il est alors possible pour l'utilisateur de synchroniser ses données avec des éditions Entreprise et Standard d'SQL Server ;
- **MSDE (Microsoft Desktop Engine)** : Cette édition est libre de droits de redistribution (redistribuable gratuitement) et est destinée aux applications ayant besoin d'exploiter et stocker

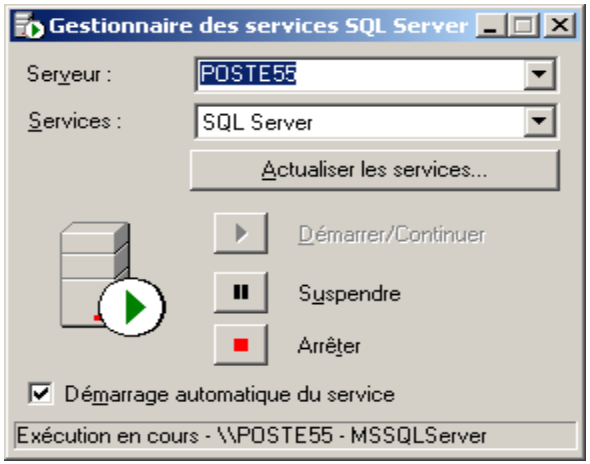
localement des bases de données. Cette édition possède des restrictions majeures notamment sur la taille maximale d'une base de données (limitée à 2Go) et du nombre d'utilisateur connecté simultanément (limité à 5).

II) Les outils de SQL Server

1. Outils graphiques

SQL Server est fourni avec une suite d'outils graphiques dédiés aux taches d'administration et d'exploitation:

- **Entreprise Manager** : Outil principal d'administration de SQL Server. Il permet d'accéder à toutes les options et toutes les fonctionnalités de SQL Server. Il se présente sous forme de composant enfichable MMC ;
- **Analyseur de requêtes** : Outil d'interrogation graphique utilisés pour analyser le plan d'une requête, analyser les informations statistiques et gérer simultanément plusieurs requêtes dans différentes fenêtres ;
- **Service Manager** : Utilitaire graphique permettant de démarrer, arrêter et suspendre des services SQL Server.



2. Outil en ligne de commande

- **Osql** : Utilitaire utilisant la connectivité ODBC pour communiquer avec SQL Server (utilisé principalement pour exécuter des fichiers de traitement par lots contenant une ou

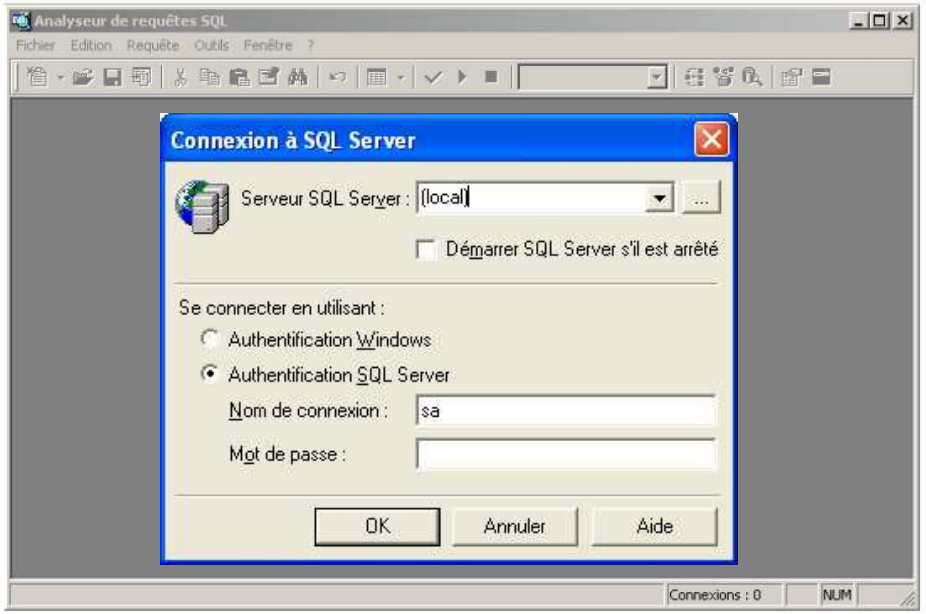
plusieurs instructions SQL). Il remplace l'utilitaire isql des versions précédentes de SQL Server.

III) Présentation de l'analyseur de requête

L'analyseur de requêtes sert à afficher sous forme graphique les instructions des requêtes et leurs résultats. Vous pouvez aussi l'utiliser pour écrire, modifier et enregistrer des scripts Transact-SQL.

Lorsque vous lancez l'analyseur de requête, il demande les informations de connexions :

- **Serveur SQL Server** : indique le nom ou l'adresse réseau du poste exécutant SQL Server. Vous pouvez utiliser « (local) » ou « . » pour désigner le poste local.
- L'**authentification Windows** qui utilise votre compte d'ouverture de session Windows.
- L'**authentification SQL Serveur** qui utilise des comptes propres à SQL Server. Dans ce mode, un login et un mot de passe sont requis. Par défaut « sa » avec un mot de passe vide.



L'analyseur de requête est composé de plusieurs fenêtres :

- **Fenêtre Explorateur d'Objet** : permets de parcourir, dans une arborescence, tous les objets de vos bases de données. La touche « F8 » permet de l'afficher ou de la masquer ;
- **Fenêtre de requêtes** : éditeur de texte pour la saisie de vos codes TSQL. Il utilise un code de couleur pour faciliter la lecture du code (Exemple: en bleu les mots clés TSQL, en rouge les chaînes de caractères, etc.). L'exécution du script peut être lancée avec la touche « F5 ». Les résultats et les messages s'affichent dans une fenêtre de texte ou dans une grille. Le code saisi peut être enregistré dans un fichier (avec l'extension .sql) pour être ouvert ultérieurement.

IV) Présentation de Osql

L'utilitaire de ligne de commande osql accepte les instructions TSQL et les envoie à SQL Server de manière interactive. Il correspond à l'analyseur de requêtes pour la ligne de commande. Il communique avec SQL Server via la bibliothèque ODBC (Open DataBase Connectivity). Les résultats sont mis en forme et affichés à l'écran.
Sa syntaxe est la suivante :

```
osql [-S serveur] [-E] [-U id_connexion]
     [-P mot_passe] [-i fichier_entrée]
     [-o fichier_sortie] [-?]
```

NB : Les options de la commande osql respectent la casse.
Utilisez l'instruction GO pour exécuter des instructions TSQL, l'instruction RESET pour supprimer toutes les instructions et l'instruction QUIT ou EXIT pour quitter l'utilitaire.

CHAPITRE 3 : CRÉATION DES OBJETS DE BASES DE DONNÉES

Avant de pouvoir stocker et manipuler des données, il est indispensable de créer une base de données et des tables. Vous pouvez ensuite créer d'autres objets pour faciliter la manipulation des données ou la gestion de la sécurité.

I) Architecture d'une base de données

1. Niveau logique

Une base de données stocke principalement un ensemble de données. Les objets de base de données vous aident à structurer vos données et à définir les mécanismes de sécurité, d'amélioration des performances et de contrôle d'intégrité des données.
Parmi ces objets de base de données, nous avons :

- **Table** : définit un ensemble de lignes ayant des colonnes associées ;
- **Contrainte** : définit les règles relatives aux valeurs autorisées dans les colonnes et constitue le mécanisme standard pour garantir l'intégrité des données ;
- **Valeurs par défaut** : définit une valeur qui est stockée dans une colonne si aucune autre valeur n'est fournie au moment de l'ajout ;
- **Index** : constitue une structure de stockage offrant un accès rapide pour l'extraction de données;
- **Vue** : permet de visualiser des données provenant d'une ou de plusieurs tables ou vues d'une même base de données ;
- **Fonction définie par l'utilisateur** : renvoie soit une valeur scalaire, soit une table. Les fonctions permettent d'encapsuler une logique exécutée fréquemment. Tout code qui doit exécuter la logique incorporée dans une fonction peut appeler cette fonction plutôt que répéter toute la logique de la fonction ;
- **Procédure stockée** : constitue un ensemble nommé d'instructions Transact-SQL pré compilées devant être exécutées ensemble ;
- **Déclencheur** : constitue une forme spéciale de procédure stockée exécutée automatiquement lorsqu'un utilisateur modifie des données d'une table ou dune vue.

2. Niveau physique

La création d'une base de données entraîne la création de 3 types de fichiers :

- **un « fichier de données principal » (.mdf)** : contient la définition des structures et les données ;
- **zéro ou plusieurs « fichier de données secondaire » (.ndf)** : contient des données uniquement ;
- **un ou plusieurs « fichier journal de transaction » (.ldf)** : enregistre toutes les informations requises pour l'annulation d'une transaction sur la base de données.

II) Gestion d'une base de données

1. Création d'une base de données

L'instruction **CREATE DATABASE** permet de créer une nouvelle base de données. Sa syntaxe est la suivante :

```
CREATE DATABASE nom_base
[ ON PRIMARY
(   NAME = nom_logique ,
    FILENAME = 'nom_fichier'
    [ , SIZE = taille_initiale ]
    [ , MAXSIZE = taille_maximale ]
    [ , FILEGROWTH = pas_dincrementation ] )]
[ LOG ON
(   NAME = nom_logique ,
    FILENAME = 'nom_fichier'
    [ , SIZE = taille_initiale ]
    [ , MAXSIZE = taille_maximale ]
    [ , FILEGROWTH = pas_dincrementation ] ) ]
```

Les arguments de cette instruction sont les suivants :

- ON PRIMARY : signifie d'utiliser le groupe de fichier « PRIMARY ». C'est le groupe de fichier par défaut ;
- Name : désigne le nom logique du fichier. Par défaut, c'est le nom de la base de données suivi de « _DATA » pour le fichier de données principal et « _LOG » pour le fichier journal de transaction ;
- Filename : désigne le chemin d'accès au fichier. Par défaut, ce fichier est crée dans le sous-dossier « Program Files\Microsoft SQL Server\MSSQL\Data\ » ;
- Size : indique la taille initiale du fichier en KB, MB ou GB. Par défaut, la taille est de 1MB pour le fichier de données

principal et de 512K pour le fichier journal de transaction. L'unité par défaut est le MB ;

- Maxsize : indique en KB, MB ou GB la taille maximale que le fichier ne devra pas excéder ;
- Filegrowth : indique le pas d'incrémentation de la taille du fichier. La valeur peut être absolue ou en pourcentage (x %) par rapport à la taille actuelle du fichier.

Pour créer une nouvelle base de données, vous devez être sur la base de données Master. Exemple :

```
USE Master
CREATE DATABASE Ecole
GO
CREATE DATABASE Test
ON PRIMARY
(  NAME = Test_Data,
   FILENAME = 'c:\test.mdf',
   SIZE = 10MB,
   MAXSIZE = 50MB,
   FILEGROWTH = 10 %)
LOG ON
(  NAME = Test_log,
   FILENAME = 'c:\test.ldf',
   SIZE = 5MB,
   MAXSIZE = 25MB,
   FILEGROWTH = 5MB )
GO
```

2. Modification d'une base de données

L'instruction **ALTER DATABASE** permet de modifier une base de données. Sa syntaxe est plus complexe mais en voici un exemple.

```
USE Master
ALTER DATABASE Test
 MODIFY NAME=Test2
GO
```

3. Suppression d'une base de données

L'instruction **DROP DATABASE** permet de supprimer une base de données. Sa syntaxe est la suivante :

```
DROP DATABASE nom_base [ , autre_base ]
```

Exemple :

```
USE Master
DROP DATABASE Ecole, Test2
GO
```

III) Gestion d'une table

1. Création d'une table

L'instruction **CREATE TABLE** permet de créer une nouvelle table. Sa syntaxe est la suivante :

```
CREATE TABLE nom_table
 ( nom_colonne   type_donnee  [contrainte] [,]
    ... )
```

Avant de créer une nouvelle table, vous devez être sur votre base de donnée utilisateur.

Exemple :

```
USE Tempdb
CREATE TABLE ABC
(  ColonneA  Int,
   ColonneB  Char(50),
   ColonneC  DateTime)
GO
```

Dans une instruction CREATE TABLE, le mot-clé **AS** permet de créer des colonnes calculées à partir des valeurs d'autres colonnes.

Exemple :

```
USE Tempdb
CREATE TABLE Note
(  NoteID Int,
   Note1  Real,
   Note2  Real,
   Moyenne as (Note1+Note2)/2,
   EtudiantID Int)
GO
```

2. Modification d'une table

L'instruction **ALTER TABLE** permet de modifier la structure d'une table. Sa syntaxe est la suivante :

```
ALTER TABLE table
{  ALTER COLUMN nom_colonne  nouveau_type
 | ADD  { CONSTRAINT  contrainte | colonne }
 | DROP { CONSTRAINT  contrainte | COLUMN column}}
```

Exemple :

```
USE Tempdb
ALTER TABLE ABC
 ADD ColonneD decimal
GO
```

3. Suppression d'une table

L'instruction **DROP TABLE** permet de supprimer une base de données. Sa syntaxe est la suivante :

```
DROP TABLE nom_table
```

Exemple :

```
USE Tempdb
DROP TABLE ABC
GO
```

IV) Implémentation des contraintes d'intégrités

Les contraintes d'intégrité permettent d'assurer des contrôles sur les données saisies dans la base de données. Il existe différentes catégories d'intégrité de données :

- **Contraintes d'entités** : définit une ligne comme une entité unique ;
- **Contrainte de domaine** : définit un intervalle ou une liste de valeur possible pour une colonne ;
- **Contrainte d'intégrité référentielle** : elle est fondée sur les relations entre les clés étrangères et les clés primaires. Elle préserve les relations définies entre les tables lorsque des valeurs, de clé étrangère, sont entrées et aussi lorsque des valeurs, de clé primaire, sont modifiées ou supprimées.

1. Contraintes d'entités

La contrainte **PRIMARY KEY** permet de définir une ou plusieurs colonnes comme la clé primaire d'une table. Elle s'utilise de la façon suivante :

```
USE Tempdb
CREATE TABLE ABC
(  ColonneA  Int  Primary Key,
   ColonneB  Varchar(30) )
GO
CREATE TABLE DEF
(  ColonneD  Int,
   ColonneE  Char(5),
   ColonneF  DateTime,
   CONSTRAINT pkDEF Primary Key(ColonneD,ColonneE)
)
GO
```

La contrainte **UNIQUE** permet de garantir qu'aucune valeur en double n'est entrée dans une colonne spécifique. Elle s'utilise de la façon suivante :

```
USE Tempdb
CREATE TABLE Facture
(  FID  Int  Primary Key,
   FCode  Char(10) Unique,
   FDate  DateTime )
GO
```

La propriété **IDENTITY** permet d'incrémenter automatiquement les valeurs numériques d'une colonne. Sa syntaxe est la suivante :

```
Identity [(valeur_initiale,pas_increment)]
```

Lorsque aucun paramètre n'est précisé, la propriété IDENTITY prend comme valeur initiale 1 et comme pas d'incrémentation 1.
Exemple :

```
USE Tempdb
CREATE TABLE Classe
(  CID   Int Identity(10,1) Primary Key,
   CNom  Varchar(20) )
GO
```

Le type de donnée **UniqueIdentifier** utilisé avec la fonction **NewId()** comme valeur par défaut permet de créer un identifiant unique pour la table (et aussi pour la base de donnée).
NB : La propriété Identity et le type de donnée UniqueIdentifier ne peuvent être utilisées qu'une seule fois dans une table.

2. Contraintes d'intégrité référentielles

La contrainte **FOREIGN KEY** appliqué à une colonne, permet de vérifier que la valeur doit exister dans la colonne référencée. Dans le cas contraire, le système renvoie un message d'erreur signalant une violation de clé étrangère. Sa syntaxe est la suivante :

```
[FOREIGN KEY  nom_colonne]
        REFERENCES table_ref ( colonne_ref ) ]
        [ ON DELETE { CASCADE | NO ACTION } ]
        [ ON UPDATE { CASCADE | NO ACTION } ]
```

Exemple :

```
USE TEMPDB
CREATE TABLE Pere
(  PID  Int  Identity  Primary Key,
   PNom  varchar(30) )
GO
CREATE TABLE Fils
(  FID  Int  Identity  Primary Key,
   FPrenoms  varchar(50),
```



```
   PID  Int  References  Pere(PID) ON DELETE
CASCADE)
GO
```

3. Contrainte de domaine

La contrainte **CHECK** permet de définir une condition que la valeur doit respecter avant d'être enregistrée. Elle s'utilise de la façon suivante :

```
USE TEMPDB
CREATE TABLE Visite
(  VID  Int  Identity  Primary Key,
   VAge  TinyInt  Check(VAge<=120),
   VJour  Varchar(20)  Check(VJour In
             ('Lundi','Mercredi','Vendredi')),
   VPoid  Real  Check(VPoid between 5 and 500))
GO
```

La contrainte **DEFAULT** permet de définir une valeur par défaut pour une colonne. Cette valeur sera utilisée lors de l'ajout d'un nouvel enregistrement, si aucune valeur n'est précisée pour cette colonne. Elle s'utilise de la façon suivante :

```
USE TEMPDB
CREATE TABLE Caution
(  CID  Int  Identity  Primary Key,
   CDate  Datetime  Default(getdate()),
   CMontant  Money  Default(3000),
   CNom  Varchar(80) )
GO
```

La contrainte **[NOT] NULL** permet de définir la possibilité pour une colonne d'accepter une valeur « NULL ». Elle est utilisée pour rendre la saisie de données obligatoire pour une colonne.
NB : La valeur « NULL » signifie qu'aucune valeur n'a été saisie dans une colonne. Elle ne doit pas être confondue avec la valeur zéro ou la chaîne de caractère de longueur zéro.
Cette contrainte s'utilise de la façon suivante :

```
USE TEMPDB
CREATE TABLE Etudiant
(  EID  Int  Identity  Primary Key,
   ENom  varchar(30) Not Null,
   EPrenoms  Varchar(80) Null  )
GO
```

CHAPITRE 4 : EXTRACTION DE DONNÉES

L'extraction ou sélection de données se base sur une requête. Une requête est une demande de données stockées dans SQL Server 2000.

I) Principes de base des requêtes

Une requête peut être émise de plusieurs manières :

- un utilisateur MS Query ou Access peut utiliser une interface utilisateur graphique (GUI, graphical user interface) pour extraire d'une ou plusieurs tables de SQL Server les données qu'il veut utiliser ;
- un utilisateur de l'Analyseur de requêtes ou de l'utilitaire osql peut émettre une instruction SELECT ;
- une application Microsoft Visual Basic peut mapper les données d'une table SQL Server à un contrôle associé, tel qu'une grille.

L'instruction SELECT extrait les données de SQL Server et les renvoie à l'utilisateur dans un ou plusieurs jeux de résultats. Un jeu de résultats est une présentation, généralement sous forme de tablueau, des données extraites par l'instruction SELECT. De même qu'une table SQL, le jeu de résultats se compose de colonnes et de lignes. La syntaxe complète de l'instruction SELECT est complexe, mais en voici les principales clauses :

```
SELECT colonne [, colonne2]
FROM table [,table2]
[WHERE Condition]
[ORDER BY colonne [ASC | DESC] [,]]
```

II) Utilisation de la clause SELECT

La clause SELECT définit la liste des colonnes du jeu de résultat. La liste est une série de colonnes de tables ou d'expressions.
Exemple:

```
SELECT FactID, FactDate, FactMontantHT*1.18
FROM Facture
GO
```

Le mot-clé « * » permet de sélectionner toutes les colonnes de la table.
Exemple :

```
SELECT *
FROM Facture
```


Le mot-clé « **IDENTITYCOL** » permet de désigner la colonne soumise à la propriété Identity.
Exemple :

```
SELECT Identitycol, FactDate
FROM Facture
```

Le mot-clé « AS » permet de définir un alias (nom plus parlant) pour une colonne. Il peut être remplacé par un simple espace.
Exemple :

```
SELECT FactID AS "Numero Facture",
       FactDate AS "Date de Facturation",
       FactMontantHT "Montant Total HT",
       FactMontantHT*1.18 "Montant Total TTC"
FROM Facture
```

III) Utilisation de la clause FROM

La clause FROM permet de spécifier les tables d'où sont extraites les données. Elle est obligatoire dans toutes les instructions SELECT sauf dans celles qui ne renvoient que des constantes.
Exemple :

```
SELECT Getdate()
GO
SELECT * FROM Article
```

IV) Utilisation de la clause WHERE

La clause WHERE permet de spécifier un critère de recherche que les lignes devront respecter pour être sélectionnées.
Exemple :

```
SELECT *
FROM Facture
WHERE FactDate >= '1/1/04'
```

La condition de recherche est une combinaison d'un ou plusieurs opérateurs logiques. En plus des opérateurs de comparaison standard, on peut aussi utiliser les suivants :

```
Expression [NOT] BETWEEN Valeur1  AND  Valeur2
Expression IS [NOT] NULL
Expression [NOT] LIKE Chaine1
```

L'opérateur LIKE permet d'effectuer des comparaisons sur des chaînes de caractères sans respecter la casse et en utilisant les caractères génériques suivants :

- % : toutes chaînes de 0 ou plusieurs caractère. Exemple: LIKE '%ordinateur%'
- _ : 1 caractère. Exemple: LIKE '_offi'
- [] : un intervalle ou une liste de caractère. Exemple: LIKE '200[0-5] ou LIKE '200[012345]
- [^] :hors de l'intervalle ou de la liste de caractère.

V) Utilisation de la clause ORDER BY

La clause ORDER BY spécifie l'ordre de tri affecté aux colonnes renvoyées dans une instruction SELECT. L'argument ASC indique un ordre de tri croissant. C'est l'ordre de tri par défaut. L'argument DESC indique un ordre de tri décroissant. La valeur « NULL » est toujours considérée comme la plus petite des valeurs.
Exemple :

```
SELECT EClasse, ENom, EDateNaissance
FROM Etudiant
ORDER BY EClasse ASC, EDateNaissance DESC
```

VI) Techniques avancées d'accès aux données

Le mot-clé DISTINCT permet d'éliminer les doublons dans le jeu de résultat d'une instruction SELECT.
Exemple :

```
SELECT DISTINCT Type
FROM Titles
/* Affiche une liste de différent type de livres
sans doublons */
```

Le mot-clé TOP permet de limiter le nombre de lignes renvoyées dans le jeu de résultat d'une instruction SELECT. Sa syntaxe est la suivante :

```
SELECT TOP N [Percent]
```

N indique le nombre absolu de lignes renvoyées ou le pourcentage si PERCENT est précisé.
Exemple :

```
SELECT TOP 10 *
FROM Titles
/* Affiche la liste des 10 premiers ouvrages de la
table Titles */
```


CHAPITRE 5 : REGROUPEMENT ET SYNTHÈSE DE DONNÉES

Ce chapitre vous permet d'acquérir les compétences nécessaires pour regrouper et synthétiser des données en utilisant des fonctions d'agrégation.

I) Utilisation des fonctions d'agrégation

Les fonctions d'agrégation (SUM, AVG, MIN, MAX et COUNT) génèrent des valeurs de synthèse dans le jeu de résultat.

1. Les fonctions Min et Max

Les fonctions Min et Max renvoient respectivement la valeur minimale et la valeur maximale d'une colonne ou d'une expression.
La syntaxe est la suivante :

```
MIN/MAX ([ALL/DISTINCT] Expression)
```

L'argument ALL indique que la fonction s'applique à toutes les valeurs. C'est l'argument par défaut. L'argument DISTINCT permet de spécifier que seules les valeurs uniques sont traitées par la fonction.
NB : Dans le cas de MIN et MAX, l'argument DISTINCT n'a aucun effet sur le résultat de la fonction.
Exemple :

```
USE PUBS
SELECT MAX(PRICE)
FROM TITLES
```

2. Les fonctions SUM et AVG

Les fonctions SUM et AVG renvoient respectivement la somme et la moyenne des valeurs d'une colonne ou d'une expression de type numérique.
La syntaxe est la suivante :

```
SUM/AVG ([ALL/DISTINCT] Expression)
```

L'argument ALL indique que la fonction s'applique à toutes les valeurs. C'est l'argument par défaut. L'argument DISTINCT permet de spécifier que seules les valeurs uniques sont traitées par la fonction. Les fonctions SUM et AVG ne tiennent pas compte des valeurs « NULL ».
Exemple:

```
SELECT SUM(NOTE)
FROM ETUDIANT
GO
```

```
SELECT AVG(NOTE)
FROM ETUDIANT
GO
```

3. La fonction COUNT

La fonction COUNT retourne le nombre d'éléments d'une colonne ou d'une expression.

La syntaxe est la suivante :

```
COUNT ([ALL/DISTINCT] Expression)
Ou
COUNT (*)
```

L'argument ALL indique que la fonction s'applique à toutes les valeurs à l'exclusion des valeurs « NULL ». C'est l'argument par défaut. L'argument DISTINCT permet de spécifier que seules les valeurs uniques sont traitées par la fonction, à l'exclusion des valeurs « NULL ». L'argument * permet de spécifier que toutes les valeurs sont prises en compte, y compris les valeurs « NULL »

Exemple :

```
USE PUBS
SELECT COUNT(*)
FROM TITLES
```

NB : Dans une instruction SELECT de base, lorsque vous utilisez une fonction d'agrégation dans une colonne, vous devez utiliser les fonctions d'agrégation pour toutes les colonnes ou alors procéder à un regroupement avec GROUP BY.

Exemple :

```
USE PUBS
SELECT MIN(Price) Minimum,
       MAX(Price) Maximum,
       AVG(Price) Moyenne
FROM TITLES
```

II) Regroupement avec GROUP BY

La clause GROUP BY est utilisée pour produire des valeurs d'agrégation regroupées selon une expression.

Syntaxe :

```
GROUP BY [ALL] Expression [,]
```

Exemple :

```
USE PUBS
SELECT Type, Min(Price), Max(Price)
FROM TITLES
GROUP BY Type
```



```
ORDER BY Type
GO
/*Cet exemple affiche la liste des type de livre
avec pour chaque type, le prix maximum, le prix
minimum et la moyenne de prix */
```

La clause ORDER BY permet d'ordonner le jeu de résultat après le regroupement des données.

La clause **HAVING** définit les critères de sélections pour la clause GROUP BY. Le critère HAVING s'applique après le regroupement des données.

La clause WHERE peut être utilisée, mais elle spécifie des critères de sélections qui s'appliquent avant le regroupement des données.

La syntaxe de HAVING est similaire à celle de WHERE à la seule différence qu'elle peut contenir des fonctions d'agrégations.

Exemple :

```
USE PUBS
SELECT Type, Min(Price), Max(Price)
FROM TITLES
GROUP BY Type
HAVING Max(Price)>1000
GO
```

Remarque : Si une colonne de regroupement contient une valeur nulle, cette ligne devient aussi un groupe dans le jeu de résultat. Si vous utilisez l'argument ALL avec GROUP BY, le jeu de résultats contiendra tous les groupes produits y compris ceux qui ne satisfont pas au critère de recherche spécifié dans la clause WHERE.

III) Synthèse avec COMPUTE

La clause COMPUTE permet de générer une valeur de synthèse apparaissant après le jeu de résultat de l'instruction SELECT.

Syntaxe :

```
COMPUTE  { {AVG | SUM | MIN | MAX | COUNT } |
Expression [,] } [BY Expression]
```

Exemple :

```
USE PUBS
SELECT Type, Price
FROM TITLES
ORDER BY Type
COMPUTE Min(Price), Max(Price) BY Type
```

NB : La clause ORDER BY devient obligatoire dès que vous utilise COMPUTE BY.

CHAPITRE 6 : JOINTURE DES TABLES

Pour retourner des données provenant de deux ou plusieurs tables, vous devez utiliser une jointure entre ces tables.

I) Généralités

Une jointure se base sur les relations entre les tables pour sélectionner les enregistrements.
Il existe deux catégories de jointures :

- les jointures internes : elles retournent les enregistrements uniquement quand les deux tables respectent les conditions de la jointure ;
- les jointures externes : elles retournent tous les enregistrements d'au moins une des deux tables.

II) Les jointures internes

Elles sont spécifiées dans la clause FROM. Dans une jointure interne, les valeurs des colonnes jointes sont comparées à l'aide d'un opérateur de comparaison, qui est généralement l'opérateur d'égalité « = ». Une jointure interne est introduite par le mot-clé **INNER JOIN**. La syntaxe est la suivante :

```
FROM TABLEA [INNER] JOIN TABLEB
ON CONDITION
```

Exemple:

```
SELECT P.nom, F.prenoms
 FROM PERE P JOIN FILS F
 ON P.numpere=F.numpere
/*Affiche la liste des fils qui ont un père en
précisant le nom du père*/
```

L'usage des alias est fortement recommandé avec les jointures. Cela permet de réduire le code et de le rendre plus lisible. Dans le cas où les tables jointes possèdent des noms de colonnes identiques, cette recommandation devient une obligation.
RQ : Une jointure interne peut être écrite en utilisant la clause WHERE. Ce type de code devra néanmoins être proscrit à cause de l'excès de travail qu'il génère pour s'exécuter.
Exemple:

```
SELECT P.nom, F.prenoms
 FROM PERE,FILS
 WHERE PERE.numpere=FILS.numpere
```



```
/*Affiche les liste des fils qui ont un père en
précisant le nom du père. Peu performant*/
```

III) Les jointures externes

Les jointures externes renvoient toutes les lignes d'au moins une des tables mentionnées dans la clause FROM et renvoient les lignes de l'autre table qui répondent à la condition mentionnée par la clause ON. La syntaxe est la suivante :

```
FROM TABLEA {LEFT|RIGHT|FULL} [OUTER] JOIN TABLEB
```

On distingue trois types de jointure externe :

- **la jointure externe gauche** : introduite par la clause **LEFT OUTER JOIN**. Elle retourne tous les enregistrements qui respectent la condition ainsi que tous les enregistrements de la table à gauche du FROM qui ne respectent pas la condition ;

Exemple:

```
SELECT P.nom, F.prenoms
 FROM PERE P LEFT JOIN FILS F
 ON P.numpere=F.numpere
/*Affiche la liste de TOUS les pères et précise le
prénom des fils lorsqu'ils existent*/
```

- **la jointure externe droite** : introduite par la clause **RIGHT OUTER JOIN**. Elle retourne tous les enregistrements qui respectent la condition ainsi que tous les enregistrements de la table à droite du FROM qui ne respectent pas la condition ;
- **la jointure externe complète** : introduite par la clause **FULL OUTER JOIN**. Elle retourne tous les enregistrements qui respectent la condition ainsi que tous les enregistrements des deux tables qui ne respectent pas la condition.

Une **jointure croisée** est un cas spécial de jointure qui permet d'obtenir le produit cartésien des tables impliquées dans la jointure. Elle est introduite par la clause **CROSS JOIN**.

CHAPITRE 7 : UTILISATION DES SOUS-REQUÊTES

Une sous-requête est une requête « SELECT » qui est imbriquée dans une autre requête. Elle peut être utilisée partout où l'on peut utiliser une expression.
Exemple:

```
SELECT nom+' ' +prenoms AS [Nom Complet]
 FROM ClasseA
 WHERE moyenne >=  (SELECT AVG(moyenne)
                    FROM ClasseA)
 ORDER BY moyenne DESC
/*Affiche par ordre décroissant de moyenne, le nom
des étudiants qui ont eu une note supérieure ou
égale à la moyenne de la classe*/
```

I) Les règles des sous-requêtes

L'utilisation d'une sous-requête obéit à plusieurs règles, dont les principales sont les suivantes :

- une sous-requête précédée d'un opérateur de comparaison ne doit retourner qu'une seule valeur sauf si elle est introduite par le mot-clé ANY, ALL ou EXISTS ;
- les clauses COMPUTE, INTO et DISTINCT ne peuvent pas être utilisées dans une sous-requête ;
- ORDER BY ne peut être utilisé qu'avec le mot-clé TOP dans une sous-requête ;
- les types de données ntext, text et image ne sont pas autorisés dans la liste de sélection d'une sous-requête.

II) Les types de sous-requêtes

En plus des opérateurs de comparaisons, les sous-requêtes peuvent être introduites par divers mots-clés, dans la clause WHERE.

1) le mot-clé IN

Le mot-clé IN permet de déterminer si une valeur donnée appartient à lune liste de valeur. Sa syntaxe est la suivante :

```
WHERE Expression [NOT] IN (sous_requete)
```

Exemple:

```
SELECT nom
 FROM Etudiant
 WHERE numetudiant IN ( SELECT numetudiant
```



```
                        FROM Boursier)
/*Affiche la liste des étudiants boursiers*/
```

2) les mots-clés ANY et ALL

Le mot-clé ALL spécifie tous les éléments retournés tandis que le mot-clé ANY spécifie l'un d'entre eux. La syntaxe est la suivante :

```
WHERE Expression Op_de_comparaison {ANY | ALL}
(sous_requete)
```

Ainsi,

>ALL (1,2,3) est équivalent à >3
>ANY (1,2,3) est équivalent à >1

Exemple :

```
SELECT nom
 FROM ClasseA
 WHERE moyenne > ALL ( SELECT moyenne
                       FROM ClasseB)
/*Affiche la liste des étudiants de la classe A qui
ont une moyenne supérieure à toutes celles des
étudiants de la classe B*/
```

RQ : le mot-clé SOME est équivalent au mot-clé ANY et existe pour la compatibilité ascendante.

3) le mot-clé EXISTS

Le mot-clé EXISTS retourne une valeur booléenne qui est vraie lorsque la sous-requête renvoie des lignes et fausse dans le cas contraire. La syntaxe est la suivante :

```
WHERE Expression [NOT] EXISTS (sous_requete)
```

Exemple :

```
SELECT nom, EXISTS(SELECT *
             FROM Stage S
             WHERE S.NumEtudiant=E.NumEtudiant)
 FROM Etudiant E
/*Affiche la liste des étudiants avec une colonne
supplémentaire indiquant si l'étudiant à déjà
effectué un stage*/
```

III) Combinaisons de résultats de requêtes

La combinaison de résultats de requêtes consiste à transformer deux ou plusieurs jeux de résultats en un seul jeu de résultat. Les jeux de résultats combinés doivent tous avoir la même structure (même nombre de colonnes et même type de données)

La combinaison se réalise à l'aide du mot-clé « UNION ». Sa syntaxe est la suivante :

```
{instruction_select_1}
 UNION [ALL]
{instruction_select_2}
 UNION [ALL]
 .
 .
 .
```

ALL indique que toutes les lignes doivent être combinées, y compris les doublons.
Exemple :

```
SELECT nom, fonction
 FROM Employe
UNION ALL
SELECT nom, fonction
 FROM Temporaire
UNION ALL
SELECT nom,'Stagiaire'
 FROM Stagiaire
/*Affiche la liste de tout le personnel, y compris
les agents temporaires et les stagiaires. Les
stagiaires n'ayant de fonction précise, la fonction
'Stagiaire' leur est affectée.*/
```


CHAPITRE 8 : Modification des données

Une fonctionnalité importante de SQL Server est sa capacité à pouvoir modifier les données stockées. Le langage SQL propose plusieurs instructions pour ajouter, modifier et supprimer les données dans une base de données.

I) Ajout de données

L'instruction « INSERT » permet d'ajouter une ou plusieurs lignes dans une table. Sa syntaxe est la suivante :

```
INSERT [INTO] nom_table [(liste_colonne)]
{VALUES ({default | expression}) | expression }
```

Exemple:

```
INSERT Etudiant (nom, prenoms)
VALUES ('KOKO','Charles')
/*Ajoute une ligne à la table Etudiant avec le nom
KOKO et le prénom Charles*/
INSERT Etudiant (nom, prenoms)
 SELECT nom, prenoms
 FROM Bachelier
 WHERE Moyenne >= 12
/* Ajoute à la table Etudiant, le nom et le prénom
des bacheliers dont la moyenne est supérieure ou
égale à 12 */
```

Dans la liste des colonnes, une colonne peut ne pas être précisées si l'une des conditions suivantes est respectée :

- elle est incrémentée automatiquement
- elle possède une valeur par défaut
- elle autorise les valeurs « Null »

Exemple :

```
CREATE TABLE Voiture (
 Numero int identity primary key,
 Immatriculation char(8) unique,
 DateArrivee datetime default('01/01/2004'))
GO
/*Crée une table nommée voiture*/
INSERT Voiture
 VALUES('3000AA01','30/01/2004')
/*Ajoute une ligne à la table voiture. On ne
précise pas de valeur pour la colonne soumise à la
propriété Identity*/
```

```
GO
INSERT Voiture(Immatriculation)
 VALUES ('3001AA01')
/*Ajoute une ligne à la table voiture en utilisant
la valeur par défaut définit pour la colonne
DateArrivee*/
```

II) Modification de données

L'instruction « UPDATE » permet de changer les valeurs d'une ou plusieurs lignes dans une table. Sa syntaxe est la suivante :

```
UPDATE nom_table
SET {nom_colonne={expression | default}[,]}
 [FROM table_source]
 [WHERE condition]
```

Exemple :

```
UPDATE Etudiant
SET nom=UCASE(nom),moyenne=moyenne+1
 WHERE classe='DSSGL'
/*Change le nom en majuscule et ajoute 1 à la
moyenne de tous les étudiants de la classe DSSGL*/
```

III) Suppression de données

L'instruction « DELETE » permet de supprimer une ou plusieurs lignes dans une table. Sa syntaxe est la suivante :

```
DELETE nom_table
 [FROM table_source]
 [WHERE condition]
```

Exemple :

```
DELETE Etudiant
 WHERE Moyenne Is Null
/*Supprime tous les étudiants qui n'ont pas de
moyenne*/
```

RQ : Pour supprimer tous les enregistrements d'une table dans SQL Server 2000, l'instruction « TRUNCATE nom_table » peut être plus efficace que « DELETE nom_table » car cette dernière utilise des mécanismes pour sauvegarder les données avant de les supprimer.

CHAPITRE 9 : Présentation des objets de programmation

Les objets de programmation vont permettre d'afficher et de manipuler des données tout en masquant la complexité de la structure de la base de données.

I) Présentation des vues

Une vue peut être considérée comme une requête enregistrée. Ainsi vous pouvez réutiliser une instruction sans avoir à la redéfinir.

1. Avantage des vues

Une vue vous permet d'effectuer les tâches suivantes :

- limiter l'accès d'un utilisateur à certaines lignes d'une table ;
- limiter l'accès d'un utilisateur à certaines colonnes d'une table ;
- joindre les colonnes de différentes tables.

2. Gestion des vues

L'instruction CREATE VIEW permet de créer une vue. Sa syntaxe est la suivante :

```
CREATE VIEW Nom_vue [(NomColonne[, n.])]
--instruction SELECT
```

Exemple :

```
CREATE VIEW TitleView
AS
 SELECT title, author
 FROM titles
GO
```

Lorsque vous créez des vues, tenez compte des restrictions énumérées ci-dessous :

- l'instruction CREATE VIEW ne peut pas inclure les clauses COMPUTE ou COMPUTE BY ;
- l'instruction CREATE VIEW ne peut pas inclure la clause ORDER BY, sauf si elle est utilisée avec une clause TOP dans l'instruction SELECT.

Une vue peut être utilisée partout où on peut une table.
Exemple :

```
SELECT * FROM TitleView
GO
INSERT TitleView VALUES ('Le Berger','Bernard')
GO
```

L'instruction ALTER VIEW permet de modifier une vue.

```
ALTER VIEW TitleView
AS
 SELECT TOP 100 PERCENT title, author
 FROM titles
 ORDER BY title
GO
```

L'instruction DROP VIEW permet de supprimer une vue.
Exemple :

```
DROP VIEW TitleView
GO
```

II) Présentation des procédures stockées

Une procédure stockée est un ensemble nommé d'instructions Transact-SQL, précompilée et stockée sur le serveur. Les procédures stockées constituent une méthode privilégié pour encapsuler les tâches répétitives afin de les exécuter efficacement. Elles prennent en charge les variables déclarées par l'utilisateur, le contrôle de flux et d'autres fonctionnalités de programmation avancées.

1. Avantage des procédures stockées

Les procédures stockées offrent de nombreux avantages :

- performance : chaque fois qu'une requête Transact-SQL est exécutée, le serveur détermine si la syntaxe est correcte puis construit un plan d'exécution avant d'exécuter la requête. Les procédures stockées sont plus performantes parce que le serveur vérifie la syntaxe à la création. La première fois que la procédure stockée est exécutée, le serveur créé le plan d'exécution et compile la procédure. Les exécutions ultérieures de cette procédure stockée seront plus rapides parce que le serveur ne fera pas de nouvelles vérifications sur la syntaxe et la construction du plan d'exécution ;
- réutilisabilité : Une fois que la procédure stockée est créée, vous pouvez l'appeler à n'importe quel moment. Ceci permet une modularité et encourage la réutilisation de votre code ;
- La simplification : Elles peuvent partager une logique d'application avec d'autres applications, contribuant ainsi à garantir la cohérence des accès aux données et de leurs modifications. Elles peuvent encapsuler une fonctionnalité

d'entreprise. Les règles et politiques de fonctionnement encapsulées dans les procédures stockées peuvent être modifiées au même endroit. Tous les clients peuvent utiliser les mêmes procédures stockées afin de garantir la cohérence des accès aux données et de leurs modifications ;

- Accès Réseau : elles contribuent à la réduction du trafic sur le réseau. Au lieu d'envoyer des centaines d'instructions Transact-SQL sur le réseau, les utilisateurs peuvent effectuer une opération complexe à l'aide d'une seule instruction, réduisant ainsi le nombre de demandes échangées entre le client et le serveur.

2. Gestion des procédures stockées

L'instruction CREATE PROCEDURE permet de créer une procédure stockée. Sa syntaxe est la suivante :

```
CREATE PROC[EDURE] nom_procedure
[@Paramètre type_données [=default] [ OUTPUT ]][,]}
AS
[BEGIN]
--instruction
--instruction
--…
[END]
```

Exemple :

```
CREATE PROC psmoyenne_etudiant
 @Etudiantnum int,
 @Moyetudiant real OUTPUT
AS
BEGIN
 SELECT @Moyetudiant = AVG(Evaluation)
 FROM Note
 WHERE Etudiantnum = @Etudiantnum
END
```

L'instruction EXECUTE permet d'exécuter une procédure. Sa syntaxe est la suivante :

```
[[EXEC[UTE]][@return_status=] nom_procedure
[[@parameter=] {value|@variable[OUTPUT]|[DEFAULT]]
```

Exemple :

```
DECLARE @Moyenne real
EXEC ps_moyenne_etudiant
 @Etudiantnum=10,
 @Moyetudiant = @Moyenne OUTPUT
```

```
PRINT @Moyenne
GO
```

III) Présentation des fonctions

Une fonction est semblable à une procédure stockée à la différence qu'elle retourne toujours une valeur. Sa syntaxe est la suivante :

```
CREATE FUNC[TION] nom_fonction (
[ @paramètre type_données [ = valeur_défaut] [,] )
RETURNS type_données_sortie
[AS]
BEGIN
--corps_fonction
RETURN expression
END
```

Exemple :

```
CREATE FUNCTION fnmoyenne (@Etudiantnum int)
 RETURNS real
AS
BEGIN
 DELCARE @Moyetudiant
 SELECT @Moyetudiant = AVG(Evaluation)
 FROM Note
 WHERE Etudiantnum = @Etudiantnum
 RETURN @Moyetudiant
END
```

Une fonction s'utilise comme une procédure stockée, mais aussi comme une table.

Exemple :

```
SELECT * From dbo.fnmoyenne (10)
GO
```

Les instructions ALTER FUNCTION et DROP FUNCTION permettent respectivement de modifier et de supprimer une fonction.

IV) Présentation des déclencheurs

Les déclencheurs ou triggers servent à étendre les mécanismes de gestion de l'intégrité à des contraintes très complexes et permettre le contrôle de saisie. Il s'agit de code déclenché lors de certains événements de la base de données. Un déclencheur est toujours rattaché à une table ou à une vue.

La syntaxe de création d'un déclencheur est la suivante :

```
CREATE TRIGGER nom_trigger
ON table_ou_vue
```



```
   {FOR | AFTER}|INSTEAD OF
   [INSERT][,][UPDATE][,][DELETE]
AS
[BEGIN)
--corps trigger
[END]
```

Un trigger peut être créé pour se déclencher à l'insertion de données (INSERT), à la suppression de données (DELETE) ou à la mise à jour (UPDATE). Il peut également être déclenché après l'exécution de la requête (AFTER et FOR pour la compatibilité ascendante), ou à la place de l'exécution de la requête (INSTEAD OF).

Les pseudo tables **INSERTED** et **DELETED** contiennent les données respectives de l'insertion ou la mise à jour (INSERTED) ou bien de la suppression (DELETED). On peut les utiliser dans des requêtes comme des tables ordinaires. La structure de ces tables est calquée sur la structure de la table ou vue sur laquelle repose le trigger.

```
CREATE TRIGGER NomEnMajuscule
       ON  TableEtudiant
       FOR INSERT, UPDATE
AS
UPDATE INSERTED
 SET NOM=UPPER(NOM)
```