M16:
SGBDR I

SQL Server 2014

Zakariaa Oulhafiane

Table des matières

I. INTRODUCTION

Outils de travail :

SQL Server 2012-2014

Visual Studio 2012

Objectif du modèle :

Une base de données relationnelle : Est un ensemble de tableau en relation entre elles.

Un SGBDR (Système de gestion de base de données relationnelles) : Est un logiciel qui permet moyennant des interfaces d'implémenter une BDR, de l'interroger (Exécuter des requêtes), de manipuler une base (Ajout/Modifie/Supprimer de données…)

Caractéristiques :

Séparation entre données et traitements.

Type de SGBDR :

Personnel (ex : Access)

Client / Serveur (ex : SQL Server, Oracle, …)

Orienté Objet

Architecture et rôles d'un SGBDR :

Architecture modulaire.

Module SQL :

LMD : Langage de manipulation des données

LDD : Langage de description de données

Gestion du disque : les données doivent être accessibles à des personnes particulières.

Sécurité des données :

Verrouillage

Transaction (Principe : Tout ou Rien)

Contrôle de l'intégrité des données.

Sauvegarde et restauration :

Stratégie de sauvegarde (Partielle/Totale)

Restauration (Incident utilisateurs/Incident matérielles)

II. Création et Suppression d'une BDR

1) Création d'une Base de données relationnelle

Avec paramètre par défaut :

```
CREATE DATABASE BasePufTDI201
```

Avec deux fichiers de données et deux fichiers journaux :

```
CREATE DATABASE BasePufTDI201
ON PRIMARY
(
        name=dataFileN1,
        filename='C:\MesBases\basePuf1.mdf',
        size=6MB,
        filegrowth=2MB
),(
        name=dataFileN2,
        filename='C:\MesBases\basePuf2.ndf',
        size=6MB,
        filegrowth=2MB,
        maxsize=1024MB
)
LOG ON
(
        name=logFileN1,
        filename='C:\MesBases\basePufLog1.ldf',
        size=6MB,
        filegrowth=2MB
),(
        name=logFileN2,
        filename='C:\MesBases\basePufLog2.ldf',
        size=6MB,
        filegrowth=2MB
)
```

2) Suppression d'une Base de données relationnelle

Syntaxe :

```
DROP DATABASE NomDeBase
```

III. Création de la structure logique

1) Création des tables

Syntaxe :

```
CREATE TABLE NomDeTable
(
        NomDuChamp1 Type ContrainteAuNiveauColonne
)
```

Exemple :

```
--Création de la table USINE
CREATE TABLE U
(
        NU int PRIMARY KEY,
        NomU varchar(20),
        Ville varchar(20)
)
--Création de la table PRODUIT
CREATE TABLE P
(
        NP int PRIMARY KEY,
        NomP varchar(20),
        couleur varchar(20) DEFAULT('BLANC'),
        poids int CHECK(poids>0)
)
--Création de la table FOURNISSEUR
CREATE TABLE F
(
        NF int,
        NomF varchar(20),
        Statut varchar(20) CHECK(statut IN ('Etranger','Nationale')),
        Ville varchar(20),
        CONSTRAINT PK_F PRIMARY KEY(NF)
)
--Création de la table PUF
CREATE TABLE PUF
(
        NP int FOREIGN KEY REFERENCES P(NP),
        NU int FOREIGN KEY REFERENCES U(NU),
        NF int REFERENCES F(NF),
        Quantité int CHECK(Quantité>=0),
        CONSTRAINT PK_PUF PRIMARY KEY(NP,NU,NF)
)
```

2) Création de table avec des données depuis une autre table

Syntaxe :

```
SELECT NP,Poids
INTO NomTable
FROM TableSource
WHERE Condition
```

3) Les Types de données

CHARACTER (ou CHAR) : Données de caractères Unicode de longueur fixe.

CHARACTER VARYING (ou VARCHAR ou CHAR VARYING) :

Données de caractères Unicode de longueur fixe de n caractères. n doit être une valeur comprise entre 1 et 4 000. La taille de stockage est de deux fois n octets.

NATIONAL CHARACTER (ou NCHAR ou NATIONAL CHAR) :

Données Unicode de longueur fixe dont la longueur maximale est égale à 4 000 caractères. Longueur par défaut = 1. La taille de stockage (en octets) correspond au double du nombre de caractères entrés.

NATIONAL CHARACTER VARYING (ou NVARCHAR) :

Données Unicode de longueur variable comptant entre 1 et 4 000 caractères. Longueur par défaut = 1. La taille de stockage (en octets) correspond au double du nombre de caractères entrés.

NUMERIC (ou DECIMAL ou DEC) :

Nombre décimal à représentation exacte à échelle et précision facultatives

INTEGER (ou INT) : Entier long

SMALLINT : Entier court

FLOAT : Réel à virgule flottante dont la représentation est binaire à échelle et précision obligatoire

REAL : Réel à virgule flottante dont la représentation est binaire, de faible précision

BIT : Données de nombre entier avec une valeur de 1 ou 0.

DATETIME : Données de date et d'heure

TIMESTAMP : Ceci est un nombre binaire unique généré automatiquement.

4) Les contraintes

Au niveau de la colonne :

PRIMARY KEY : clef primaire

NOT NULL : la saisie est obligatoire

DEFAULT(Value) : la valeur par défaut

CHECK(Condition) : la saisie va être valide si la condition est valide

UNIQUE : ce champ ne répète pas

FOREIGN KEY Reference Table(colonne) : clef étrangère

Au niveau de la table :

CONSTRAINT nomContrainte contrainte

5) Modifier la structure d'une table

Ajouter la colonne DateLivraison :

```
ALTER TABLE PUF
    ADD DateLivraison datetime
```

Ajouter la colonne effectif à la table usine de type float :

```
ALTER TABLE U
    ADD effectif float
```

Modifier le type de la colonne effectif dans la table usine :

```
ALTER TABLE U
    ALTER COLUMN effectif int
```

Renommer la colonne effectif :

1er Méthode :

Supprimer la colonne effectif et ensuite ajouter la nouvelle colonne

(Mais les données seront perdues !)

2ème Méthode :

Utiliser **sp_rename** (les données ne seront pas perdues)

```
EXEC sp_rename 'U.effectif','NouveauNom','column'
```

Supprimer la colonne effectif de la table usine :

```
ALTER TABLE U
    DROP COLUMN effectif
```

Ajouter une contrainte :

```
ALTER TABLE PUF
    ADD CONSTRAINT dateParDefaut DEFAULT(getdate()) FOR DateLivraison
ALTER TABLE F
    ADD CONSTRAINT verifVille CHECK(Ville in ('casa','rabat'))
```

Note :

Il faut que la table F ne contienne pas d'autres champs que casa et rabat dans la colonne Ville

Modifier une contrainte :

Tu es obligé de la supprimer et ajouter à nouveau la nouvelle contrainte.

Supprimer une contrainte :

```
ALTER TABLE PUF
    DROP CONSTRAINT dateParDefaut
```

Désactiver et activer une contrainte :

Désactivation :

```
ALTER TABLE PUF
    NOCHECK CONSTRAINT PK_PUF_F
```

Activation :

```
ALTER TABLE PUF
    CHECK CONSTRAINT PK_PUF_F
```

6) Intégrité référentielle

L'intégrité référentielle préserve les relations définies entre les tables lors de l'insertion ou de la suppression de lignes. Dans SQL Server, l'intégrité référentielle est fondée sur les relations entre les clefs étrangères et les clefs primaires ou entre les clefs étrangères et les clefs uniques, via les contraintes FOREIGN KEY et CHECK. Elle garantit la cohérence des valeurs de clefs entre les tables. Ce type de cohérence impose qu'il n'y ait aucune référence à des valeurs inexistantes et que, si la valeur d'une clef change, toutes les références qui y sont faites soient modifiées en conséquence dans l'ensemble de la base de données.

Une intégrité référentielle n'autorise pas la saisie de la valeur dans un champ valeur si elle n'existe pas dans le champ référence.

Lorsque vous mettez en application l'intégrité référentielle, SQL Server interdit aux utilisateurs d'effectuer les opérations suivantes :

Ajouter ou modifier des lignes dans une table connexe lorsqu'il n'y a aucune ligne associée dans la table primaire.

Changer des valeurs dans une table primaire qui engendreraient des lignes orphelines dans une table connexe.

Supprimer des lignes dans une table primaire si des lignes connexes correspondantes existent.

Contraintes d'intégrité référentielle en cascade :

Les contraintes d'intégrité référentielle en cascade définissent les actions exécutées par SQL Server lorsqu'un utilisateur tente de supprimer ou de mettre à jour une clef vers laquelle pointent des clefs étrangères existantes.

[ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]

[ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]

Exemple :

```
CREATE TABLE Exemple
(
        --Suppression en cascade
        NF int FOREIGN KEY REFERENCES F(NF) ON DELETE CASCADE
        --Update en cascade
        NF int FOREIGN KEY REFERENCES F(NF) ON UPDATE CASCADE
        --Mettre la valeur NULL en suppression
        NF int FOREIGN KEY REFERENCES F(NF) ON DELETE SET NULL
)
```

IV. Manipulation des données

1) Insertion

Syntaxe :

```
INSERT INTO NomTable VALUES (.. , .. , ..)
INSERT INTO NomTable VALUES (.. , .. , ..),(.. , .. , ..)

-- Ajouter juste dans des colonnes
INSERT INTO NomTable (col3, col2) VALUES (.. , .. , ..)


-- Ajouter une liste depuis une autre table
INSERT INTO TableDestination SELECT Mle, Nom
                             FROM Stagiaire
```

2) Suppression

Syntaxe :

```
DELETE NomTable
FROM lesTablesDeJointure
WHERE condition
```

Note : Aussi il y a la clause **TRUNCATE** pour la suppression.

3) Mise à jour des données

Syntaxe :

```
UPDATE NomTable
SET colonne1=nouvelleVal, colonne2=nouvelleVal2
FROM lesTableDeJointure
WHERE condition
```

V. Les Fonctions

1) Fonctions sur les chaines de caractères

LIKE:

```
--Les villes qui commence par la lettre a
WHERE ville LIKE 'a%'
```

% : Chaine de caractère quelconque

_ : Un caractère Quelconque

Exemple :

```
--Un nom qui se termine par z
... WHERE nom LIKE '%z'
--Un nom qui contient la lettre a et e
... WHERE nom LIKE '%a%' AND '%e%'
... WHERE nom LIKE '%a%e%' OR '%a%e%'
--Un nom commence par un caractère entre f et k.
... WHERE nom LIKE '[F-K]%'
--Un nom commence par e, x ou t.
... WHERE nom LIKE '[E, X, T]%'
--La liste des nom qui ne se commence ni par a ni par c ni par e.
... WHERE nom LIKE '[^ACE]%'
```

LEN:

```
SELECT NF,NomF,'Longeur du nom'=LEN(NomF)
FROM F
```

SUBSTRING:

```
SELECT NF, NomF, 'Longeur du nom'=LEN(NomF)
FROM F
WHERE SUBSTRING(NomF, LEN(NomF), 1) = 'a'
```

Concatenation:

```
SELECT 'info Fournisseur'=CAST(nf AS VARCHAR(20))+' ; '+nomF
FROM F
```

Conversion des de caractère:

CONVERT:

```
SELECT 'info Fournisseur'=CONVERT(VARCHAR(20),nf)+' ; '+nomF
FROM F
```

CAST:

```
SELECT 'info Fournisseur'=CAST(nf AS VARCHAR(20))+' ; '+nomF
FROM F
```

RIGHT() :

```
SELECT RIGHT('BonjourSQLServer', 12) => 'ourSQLServer'
```

LEFT() :

```
SELECT LEFT('BonjourSQLServer', 12) => 'BonjourSQLSe'
```

REPLACE() :

```
REPLACE(nomColonne, 'Phrase à modifier', 'La nouveau phrase')
```

REPLICATE() :

```
REPLICATE('Char', NbrDeFois)
```

Exemple :

```
REPLICATE('Az',3) => 'AzAzAz'
```

2) Fonctions sur les dates

Questions :

a) Afficher les usines approvisionnées en 2016.

```
SELECT *
FROM PUF
WHERE YEAR(DateLivraison) = 2016
```

b) Afficher les usines approvisionnées le mois septembre.

```
SELECT *
FROM PUF
WHERE MONTH(DateLivraison) = 9
```

c) Afficher les usines approvisionnées un mois de juin 2016.

```
SELECT *
FROM PUF
WHERE YEAR(DateLivraison) = 2016
        AND
        (MONTH(DateLivraison) = 6)
```

d) Quelle le jour de votre naissance.

```
SET DATEFORMAT dmy;
SELECT DATENAME(dw, '02/02/1990');          => Friday
SELECT DATENAME(weekday, '02/02/1990');     => Friday
SELECT DATENAME(mm, '02/02/1990');          => February
SELECT DATENAME(month, '02/02/1990');       => February
```

e) Donner le nombre de jour, le nombre de mois entre votre date de naissance et aujourd'hui

```
SET DATEFORMAT dmy;
SELECT DATEDIFF(d, '02/02/1990', '19/10/2016')   => 9756
SELECT DATEDIFF(m, '02/02/1990', '19/10/2016')   => 320
```

f) Quelle est la date d'un rendez-vous qui sera programmé dans 15 jours.

```
SELECT DATEADD(day, 15, getdate())
```

g) Quelle est la date d'un rendez-vous qui sera programmé d'ici 2 mois.

```
SELECT DATEADD(month, 2, getdate())
```

h) Quelle est la date d'un rendez-vous qui sera programmé d'ici 2 semaines.

```
SELECT DATEADD(week, 2, getdate())
```

3) Fonctions d'agrégation

Les fonctions d'agrégation ne doivent pas être imbriquées.
Les calculs s'appliquent sur les valeurs not nulle.

Somme :

```
Select SUM(POIDS)'Total Poids'
FROM p
```

Moyenne :

```
Select AVG(POIDS)
FROM P
```

MAX :

```
Select MAX(POIDS)
FROM P
```

MIN :

```
Select MIN(POIDS)
FROM P
```

COUNT :

```
SELECT COUNT(NP)
FROM PUF

SELECT COUNT(DISTINCT NP)
FROM PUF
```

```
SELECT COUNT(DISTINCT Couleur)'Nb de couleurs'
FROM P
```

Regroupement :

DISTINCT :

Permet de regrouper des lignes afin d'éviter les doublons

GROUP BY :

Permet d'effectuer des opérations grâce aux fonctions d'agrégation

Exemple :

```
--Le total de quantités livrées par produit
SELECT NP, SUM(Quantité)
FROM PUF
GROUP BY NP
```

Le regroupement et la clause HAVING :

Comme la clause **WHERE** mais s'applique après le regroupement.

4) La Fonction EXISTS

C'est une fonction booléenne qui renvoie :

TRUE, Si la requête renvoie au moins une ligne.

FALSE, Si la requête ne renvoie aucune ligne.

Syntaxe :

```
Exists(Requête SELECT)
```

Exemple :

```
--Afficher les usines qui ne sont pas approvisionné par le fournisseur N°1.
SELECT NU
FROM U
WHERE EXISTS(SELECT *
             FROM PUF
             WHERE (NF=1) AND (PUF.NU=U.NU)
             )
```

Note : Cette requête dans le paramètre de la fonction EXIST est synchrone.

5) La fonction ISNULL() et IS NULL :

IS NULL :

```
--Les usines dont la ville n'est pas renseignée.
SELECT *
FROM U
WHERE Ville IS NULL
```

X IS NULL ==> Renvoi **TRUE** si **X** est Nul

==> Renvoi **FALSE** si **X** n'est pas Nul

ISNULL() :

Syntaxe :

ISNULL(Expression, Valeur) => Renvoi Valeur si l'expression est nulle.

Exemple :

ISNULL(Commission, 0)

Si la commission est nulle alors la valeur retourné est 0.

6) La fonction NULLIF :

Syntaxe :

NULLIF(exp1, exp2)

Renvoie **NULL** Si l'expression 1 est égale à **l'expression 2**

Sinon renvoie **l'expression 1**

Exemple :

```
SELECT NF,NomF,Ville,'TypeF'=ISNULL(NULLIF(Ville,'CASA'), 'F.Casablancais')
```

VI. Tri

Clause ORDER BY :

```
SELECT NF,NomF,PrenomF
FROM F
ORDER BY NomF,PrenomF DESC
```

Note : La seule clause qui accepte les alias

```
SELECT NF, NomF, PrenomF, X=Status
FROM F
ORDER BY X
```

Clause TOP :

```
--Sélectionner les 3 premiers
SELECT TOP 3 NF, NomF
FROM F
ORDER BY nomF
```

VII. Les Tables temporaires

Il Faut commencer la table par # pour créer une table temporaire

Exemple :

```
CREATE TABLE #TDI
(
        Mle int PRIMARY KEY,
        Nom VARCHAR(20)
)
```

Note :

Le chemin des tables temporaires se trouve dans la base : System Databases => tempdb

Elles disparaissent lorsqu'on se déconnecte du serveur

VIII. Les vues

Avantages :

Objet Stocké

Similaire à une table

Création :

```
CREATE VIEW NomView
AS
        SELECT (Requête SQL)
```

Exemple :

```
CREATE VIEW VueFournisseur
AS
        SELECT NF,PUF.NU 'Numéro Usine', U.Ville 'VilleUsine'
        FROM PUF,U
        WHERE PUF.NU=U.NU

--Liste des fournisseurs approvisionnent à une usine de CASA depuis la vueFournisseur
SELECT *
FROM VueFournisseur
WHERE VilleUsine='CASA'

--Créer une vue usineCasa
CREATE VIEW UsineCasa
AS
        SELECT NU, NomU, NbrSalarie 'Effectif'
        FROM U
        WHERE Ville='CASA'
```

Note : Il y a des vues system qui commence par le mot clef **sys**

Pour effectuer une Mise à jour à travers les vues il faut que :

La vue soit simple : - Sur une seule table

- Pas de GROUP BY

-PAS de fonctions d'agrégations

Exemples :

```
--Insertion à travers une vue
INSERT INTO UsineCasa VALUES(1002,'OULHAFIANE', 300)
```

```
--Mise à jour à travers une vue

UPDATE UsineCasa
SET VilleUsine='Rabat'
WHERE NF=1002

--Suppression à travers une vue

DELETE UsineCasa
WHERE NF=1002
```

Modification une vue :

```
ALTER VIEW UsineCasa
AS
    SELECT NU, NomU, NbrSalarie 'Effectif'
    FROM U
    WHERE Ville='CASA'
    WITH CHECK OPTION
```

Note :

WITH CHECK OPTION : Avant toute mise à jour, la condition **WHERE** de la vue doit être vraie

Suppression d'une vue :

```
DROP VIEW UsineCasa
```

IX. Les Transactions

Syntaxe :

```
BEGIN TRANSACTION

    Requête SQL
```

Commit => Validation

Rollback => Annulation

Note :

CREATE => Clause Auto comité.

DROP => Clause Auto comité.

Mais sous SQL-SERVER n'est pas possible d'ajouter une clause auto comité dans une transaction.

X. Catégorie de droits (CASE)

Syntaxe :

Case ...

WHEN ... **THEN**

END

Exemple (Type 1) :

```
--Afficher :
--'Objet Lourd' Si poids > 100.
--'Objet Semi Lourd' Si poids compris entre 50 et 100.
--'Objet Leger' si poids < 50
SELECT NP, POIDS, 'Nature poids' =
                Case
                        WHEN POIDS>100 THEN 'Poids Lourd'
                        WHEN POIDS BETWEEN 50 and 100 THEN 'Poids Semi Lourd'
                        WHEN POIDS IS NULL THEN 'à Saisir'
                        ELSE 'Poids Leger'
                 END
FROM P
```

Exemple (Type 2) :

```
--Afficher la taxe de 10% Si le fournisseur est National et de 15% si le fournisseur
--est Etranger
--Si Statut est null la taxe est 10%
SELECT NF, NomF, Taxe =
        CASE UPPER(Statut)
                WHEN 'NATIONAL' THEN '10%'
                WHEN 'ETRANGER' THEN '15%'
                ELSE '10%'
        END
FROM F
```

XI. Index

On indexe les colonnes de tables pour accélérer la recherche.

Syntaxe :

```
CREATE INDEX nomIndex ON nomTable(Colonne)
```

Exemple :

```
CREATE INDEX IX_VD_VOL ON VOL(VD)
```

Suppression d'un index :

```
DROP INDEX IX_VD_VOL ON VOL

DROP INDEX VOL.IX_VD_VOLL
```

XII. Identity (Auto increment)

Syntaxe :

```
IDENTITY(Valeur Initiale, le pas)
```

Exemple :

```
CREATE TABLE CLIENT
(
        IDClient int IDENTITY(1,1) PRIMARY KEY
        NomClient VARCHAR(25)
)
```

Remarque :

On a le droit d'ajouter un seul colonne auto incrément.

Attention à la suppression et les relations.

Note :

Connaitre la valeur courante :

```
SELECT 'Valeur Courante'=IDENT_CURRENT('CLIENT')
```

Connaitre la valeur du pas :

```
SELECT 'Valeur du pas'=IDENT_INCR('CLIENT')
```

Connaitre la valeur du départ :

```
SELECT 'Valeur Départ'=IDENT_SEED('CLIENT')
```

XIII. Les synonymes

Syntaxe :

```
CREATE SYNONYM NomSynonyme FOR NomTable
```

Exemple :

```
CREATE SYNONYM FE FOR FournisseurEtrangers
```

Suppression d'un synonyme :

```
DROP SYNONYM NomSynonyme.
```

XIV. Les rules

S'applique sur une seule colonne.

Syntaxes :

Création :

CREATE RULE NomRule

AS Condition

Bind :

EXEC SP_BINDRULE 'NomRule', 'NomTable.NomColonne'

Unbind :

EXEC sp_unbindrule 'NomTable.NomColonne', 'NomRule'(Facultatif)

Suppression d'un rule :

DROP RULE NomRule

Exemple :

```
-- Création de la règle
        CREATE RULE Sup50
        AS @colonne >= 50
--Lier une colonne à cette règle
        EXEC SP_BINDRULE 'Sup50','P.Poids'
--Supprimer la liaison entre la règle est la colonne Poids de la table POIDS
        EXEC sp_unbindrule 'P.Poids'
--Supprimer la règle
        DROP RULE Sup50
```

Remarque :

L'application de la règle ne s'applique pas sur les anciennes valeurs.

Avant de supprimer une règle il faut supprimer tous les liaisons avec cette règle.

XV. La gestion des utilisateurs (SQL)

Création d'un login sur le serveur :

EXEC SP_ADDLOGIN NomLogin, Password, BaseParDefaut

Création d'un utilisateur sur une data base :

Use DatabaseName

CREATE USER NomUser

FOR LOGIN NomLogin

Accorder des permissions :

GRANT NomPermission **ON** NomTable **To NomUtilisateur**

GRANT NomPermission **To NomUtilisateur**

Enlever des permissions :

REVOKE NomPermission **ON** NomTable **To NomUtilisateur**

REVOKE NomPermission **TO NomUtilisateur**

Interdire des permissions :

DENY NomPermission **ON** NomTable **To NomUtilisateur**

DENY NomPermission **TO NomUtilisateur**

Questions :

1)En tant que ADMIN Créer un Login avec une base de données par défaut :

```
--Créer un Login LOGTDI
EXEC SP_ADDLOGIN LOGTDI, QWERTY, PufBaseTDI201
```

2)Connectez-vous au serveur via le login (Q1).

Remarque : On ne peut pas accéder à la base de données par défaut.

3)En tant que ADMIN Créer un Utilisateur sur la base de données par défaut.

```
--Créer un User pour le login LOGTDI
Use PufBaseTDI201
CREATE USER UserTDI201
```

```
FOR LOGIN LOGTDI
```

4)Connectez-vous au serveur via le login et vérifier un select sur une table.

Remarque : Les objets inaccessibles

5)Connectez-vous en tant que ADMIN et accorder à l'utilisateur avec GRANT
Un Select sur la table U.
Un Insert sur la table P.
Un Insert sur la table F.
Un Delete sur la table P.

```
GRANT SELECT ON U TO UserTDI201
GRANT INSERT ON P TO UserTDI201
GRANT INSERT ON F TO UserTDI201
GRANT DELETE ON P TO UserTDI201
```

6)Connectez-vous en tant qu'Utilisateur et vérifier les permissions de la Q5.

Les permissions sont accordées.

7)Connectez-vous en tant que ADMIN et enlever les permissions avec REVOKE.

```
REVOKE SELECT ON U FROM UserTDI201
REVOKE INSERT ON P FROM UserTDI201
REVOKE INSERT ON F FROM UserTDI201
REVOKE DELETE ON P FROM UserTDI201
```

XVI. Les roles

Syntaxe :

```
CREATE ROLE NomRole
```

Questions :

1-Créer un rôle TDI201 qui regroupe les privilèges suivant :
SELECT ON U
INSERT ON P
DELETE ON F

```
CREATE ROLE RoleTDI201
GRANT SELECT ON U TO RoleTDI201
GRANT INSERT ON P TO RoleTDI201
GRANT DELETE ON F TO RoleTDI201
```

2-Affecter le rôle à l'utilisateur crée précédemment (L'utilisateur doit être membre du rôle).

```
EXEC sp_addrolemember 'RoleTDI201','UserTDI201'
```

3-Connecter tant que UserTDI201 et vérifier le rôle affecté.

Le rôle a été bien affecté.

4-Affecter le privilège INSERT ON U to RoleTDI201 et vérifier que l'utilisateur UserTDI201 à hériter le privilège INSERT ON U.

Oui l'utilisateur a le privilège **INSERT ON U.**

5-Donner le privilège CREATE TABLE à le rôle RoleTDI201.

```
GRANT ALTER ON Schema::DBO To RoleTDI201

GRANT CREATE TABLE TO RoleTDI201
```

Remarque :
Il est nécessaire d'ajouter le privilège ALTER ON Schema::DBO
à le rôle RoleTDI201

6-Supprimer le rôle RoleTDI201.

```
DROP ROLE NomRole
```

Attention : Il faut que le rôle soit vide de membres.

```
EXEC sp_droprolemember 'RoleTDI201', 'UserTdi201'
```

XVII. Attacher / Détacher une Base de données

Attention : Vérifier / Connaitre le chemin de base crée.

Chemin par défaut : Dossier d'installation...\Instance\DATA

Syntaxe :

```
sp_dettach_db mabase, 'FichierMDF', 'FichierNDF', ..., 'FichierLDF'

sp_attach_db mabase, 'FichierMDF', 'FichierNDF', ..., 'FichierLDF'
```

XVIII. Import / Export de données

En peut importer des données externes à l'aide de **SQL SERVER IMPORT AND EXPORT DATA.**

XIX. Sauvegarde / Restauration

```
--Ajouter une unité de sauvegarde
exec sp_addumpdevice 'Disk', 'UnitSave2', 'C:\sauvegardeBD\monUnit2.bak'

--Faire une sauvegarde totale
BACKUP DATABASE PufBaseTDI201 TO UnitSave

--Faire une sauvegarde différentiel
BACKUP DATABASE PufBaseTDI201 TO UnitSave WITH DIFFERENTIAL

--Faire une restauration total et ensuite une restauration depuis un fichier n°15 (différentiel)
RESTORE DATABASE tpsauvegarde
FROM UnitSave
WITH NORECOVERY

RESTORE DATABASE tpsauvegarde
FROM UnitSave
WITH FILE = 15, RECOVERY;
```

XX. Sous-Totaux (ROLLUP)

Exemple :

```
SELECT Ville, NomP, sum(Quantite)
FROM PUF JOIN F ON PUF.NF=F.NF JOIN P ON PUF.NP=P.NP
GROUP BY Ville, NomP WITH ROLLUP
ORDER BY Ville DESC
```

XXI. Les views In-Line

Exemple :

```
CREATE VIEW View1
AS
        SELECT ISNULL(A.Num_Client,B.Num_Client), TotalSolde, TotalEmprunt
        FROM (SELECT Num_Client,SUM(Solde) 'TotalSolde'
                FROM COMPTE
                GROUP BY Num_Client) AS A
        FULL JOIN
                (SELECT Num_Client,SUM(Montant) 'TotalEmprunt'
                FROM EMPRUNT
                GROUP BY Num_Client) AS B
        ON A.Num_Client=B.Num_Client
```

XXII. Créer une table avec un champ calculé

Exemple :

```
CREATE TABLE Facture
(
        NumFacture int PRIMARY KEY,
        RefPdt int,
        PU money,
        Qte int,
        Montant AS PU*Qte
)
```

XXIII. Fusion de tables (MERGE)

Rôle :

Effectue des opérations d'insertion, de mise à jour ou de suppression sur une table cible selon les résultats d'une jointure avec une table source.

Par exemple, vous pouvez synchroniser deux tables en insérant, mettant à jour ou supprimant des lignes dans une seule table selon les différences trouvées dans l'autre table.

Syntaxe :

```
MERGE INTO table1
USING table_reference
ON (conditions)
WHEN MATCHED THEN
        UPDATE SET table1.colonne1 = valeur1, table1.colonne2 = valeur2
        DELETE WHERE conditions2
WHEN NOT MATCHED THEN
        INSERT (colonnes1, colonne3)
        VALUES (valeur1, valeur3)
```

MERGE INTO : permet de sélectionner la table à modifier
USING et **ON** : permet de lister les données sources et la condition de correspondance
WHEN MATCHED : permet de définir la condition de mise à jour lorsque la condition est vérifiée
WHEN NOT MATCHED : permet de définir la condition d'insertion lorsque la condition n'est pas Vérifiée

Exemple :

```
--Création d'une table Tsource
        select Empno, ename, hiredate
        into Tsource
        from Employes
--Création d'une table Tcible
        select Empno AS NumSalarie, Ename AS NomSalarie, Hiredate AS DateRec
        into Tcible
        FROM Tsource
--Insérer les employés dont le numéro n'existe pas dans la source
--Faire une mise à jour de la colonne DateRec pour ceux qui existent dans les deux tables
MERGE INTO Tcible
USING Tsource
ON (Empno = NumSalarie)
WHEN MATCHED THEN
        UPDATE SET DateRec=GETDATE()
WHEN NOT MATCHED THEN
        INSERT (NumSalarie, NomSalarie, DateRec) VALUES (Empno, Ename, Hiredate);
```