

Réalisation de scripts 2^e partie

Définition

Instructions

Exemples

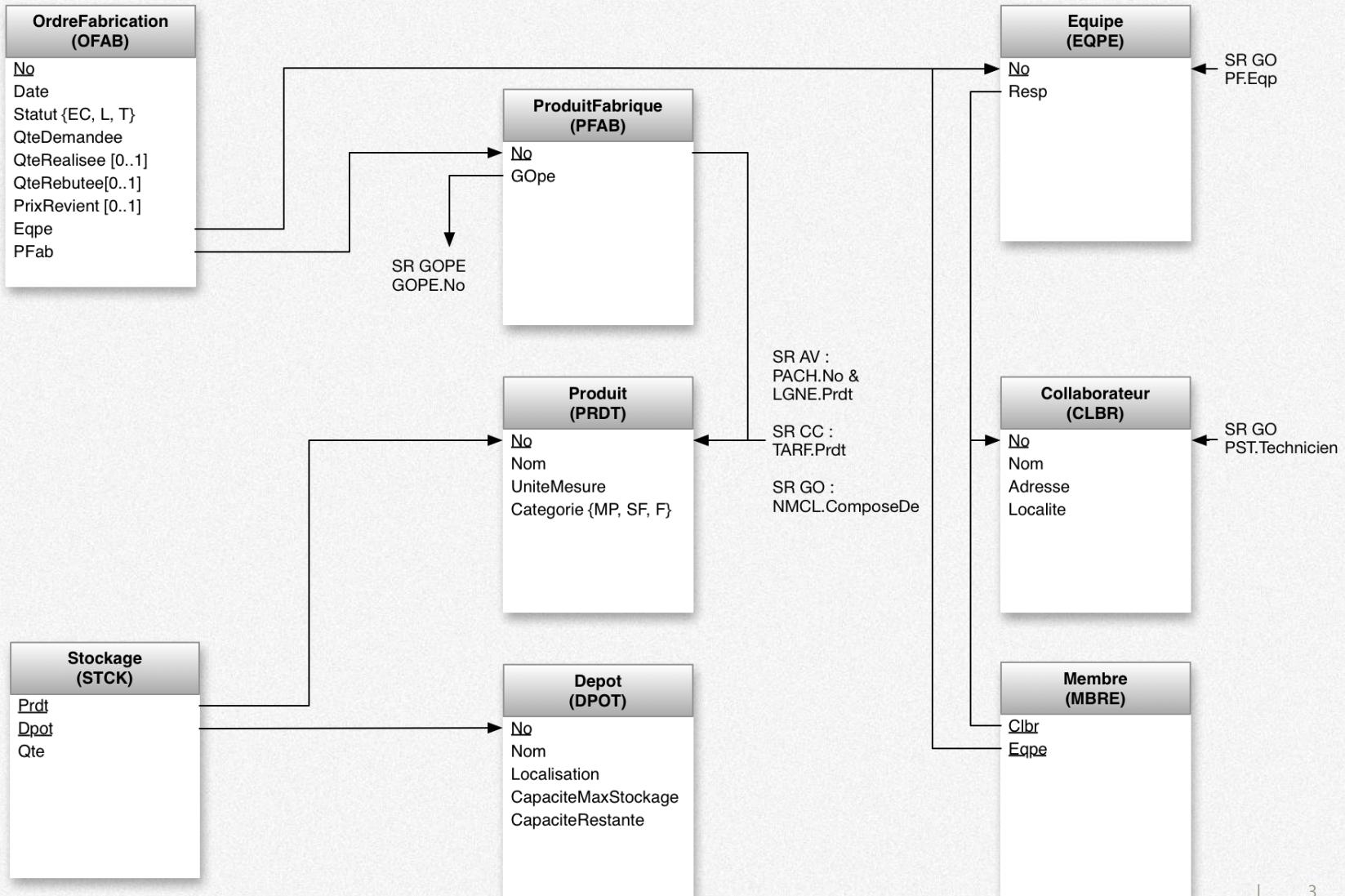
Agenda

Lors de cette session nous allons explorer un peu plus avant le langage T-SQL et ce qu'il permet de faire :

1. Création de contraintes (default, unique et check)
2. Modification de structures simples
3. Mise à jour de données
4. Suppression de données et de structures simples
5. Restrictions et contraintes de clé(s) étrangère(s)

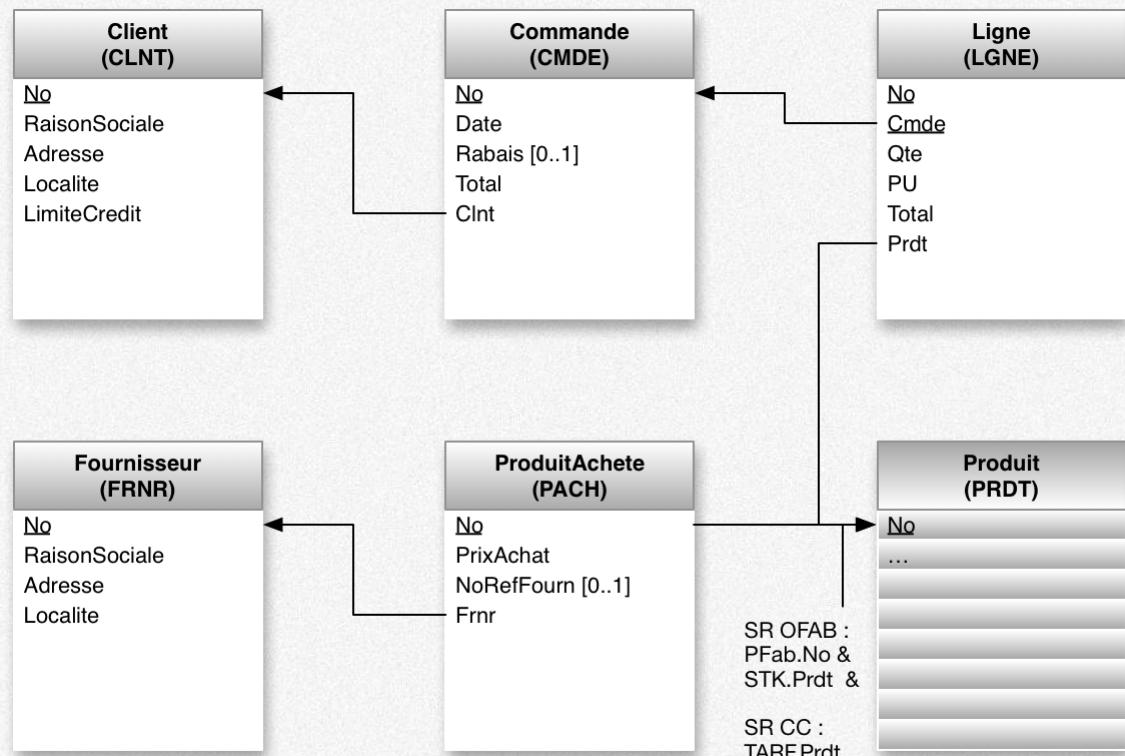
Problème

La figure ci-dessous présente le schéma relationnel correspondant à la partie *Ordre de fabrication* du problème fil rouge.



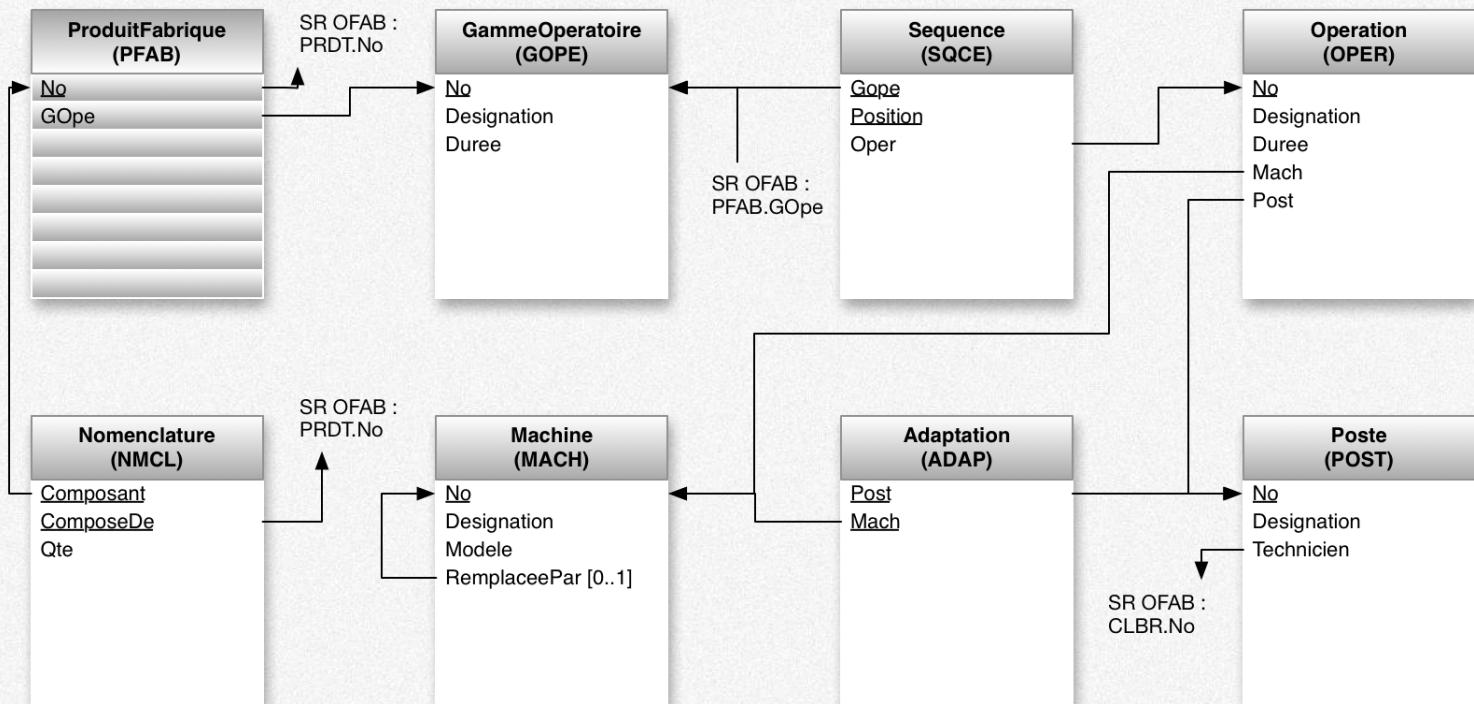
Problème

La figure ci-dessous présente le schéma relationnel correspondant à la partie *Achat-vente* du problème fil rouge.



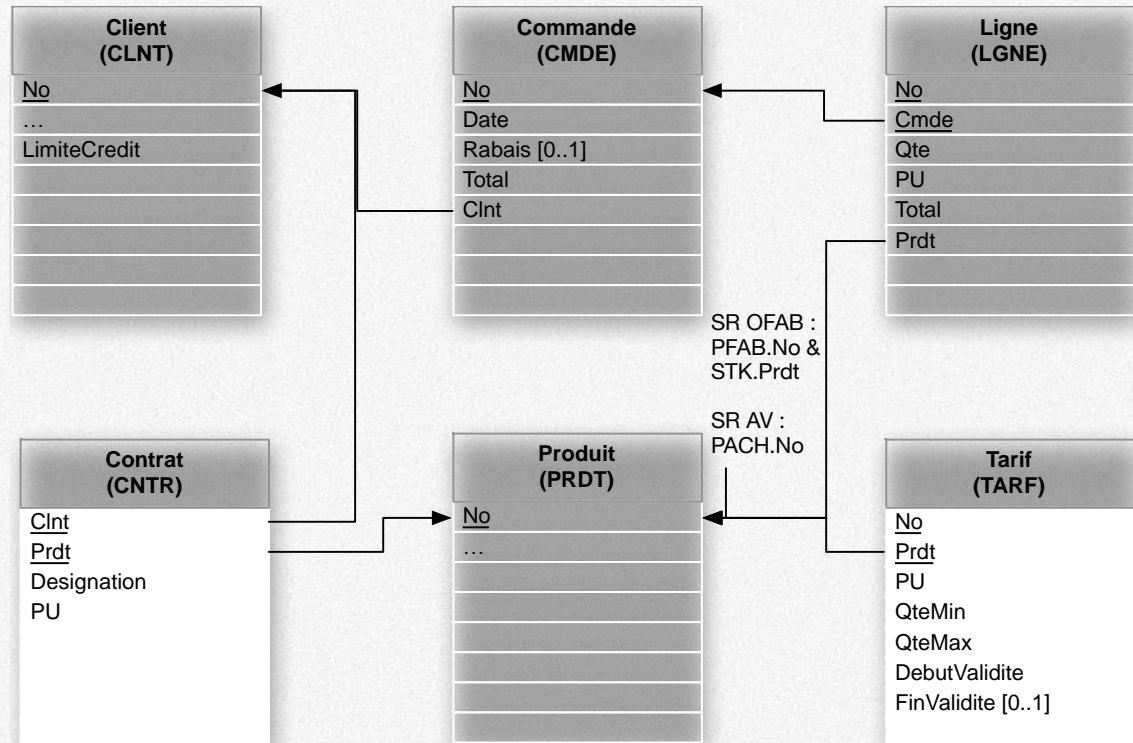
Problème

La figure ci-dessous présente le schéma relationnel correspondant à la partie *Gamme opératoire* du problème fil rouge.



Rappel

La figure ci-dessous présente le schéma relationnel correspondant à la partie *Tarif et Contrat* du problème fil rouge.



Contrainte DEFAULT

Lors de la création d'une table, il est possible de définir un certain nombre de contraintes pour chaque colonne, afin de restreindre les valeurs qui peuvent y être insérées.

Nous avons déjà vu la contrainte native `NOT NULL`, à utiliser lorsqu'une colonne ne devrait pas pouvoir recevoir la valeur `NULL`.

Il existe d'autres contraintes relatives aux colonnes permettant d'assurer la qualité des données insérées dans la base de données.

DEFAULT

Indique la valeur par défaut que la colonne contiendra pour une ligne **lorsqu'aucune** valeur n'est donnée lors de l'insertion.

OrdreFabrication (OFAB)
<u>No</u>
Date
Statut {EC, L, T}
QteDemandee
QteRealisee [0..1]
QteRebutee [0..1]
PrixRevient [0..1]
Egpe
PFab

```
-- Par défaut, un ordre de fabrication (OFAB) a
-- un statut « En cours »
CREATE TABLE OFAB (
    [...]
    Statut VARCHAR(10) NOT NULL
    [...]
);

ALTER TABLE OFAB ADD
CONSTRAINT OFAB_Statut_Def DEFAULT('EC') FOR Statut
```

Contrainte DEFAULT

Lorsqu'une contrainte de type DEFAULT a été définie sur une colonne, la valeur par défaut sera insérée dans la colonne si aucune valeur n'est indiquée lors de l'insertion.

Si vous écrivez une requête `INSERT` sans lister les colonnes, vous devrez manuellement indiquer que la colonne doit prendre sa valeur par défaut en utilisant le mot-clé `DEFAULT` :

```
-- La troisième colonne est le Statut, avec une valeur par défaut de 'EC' :  
INSERT INTO OFAB VALUES (12, '2017-09-26', DEFAULT, 2700, NULL, NULL, NULL, 4, 15);
```

Si la requête écrite liste les colonnes concernées, il suffit d'omettre la colonne possédant une valeur par défaut pour qu'elle prenne justement cette valeur :

Les colonnes QteRealisee, QteRebutee et PrixRevient ont aussi été omises. Elles sont optionnelles et vaudront donc NULL pour cette ligne.

```
-- La colonne Statut a été omise et vaudra donc  
-- 'EC' pour cette ligne.  
INSERT INTO OFAB (No, Date, QteDemandee, Eqpe, PFab)  
VALUES (12, '2017-09-26', 2700, 4, 15);
```

Contrainte UNIQUE

Cette contrainte empêche que deux lignes d'une même table ne possèdent la même valeur pour la colonne sur laquelle elle s'applique.

Poste (POST)
No Designation
Technicien

```
-- La Designation d'un Poste devrait être unique
CREATE TABLE POST (
    [...]
    Designation VARCHAR(30) NOT NULL
    [...]
);

ALTER TABLE POST ADD
CONSTRAINT POST_Designation_Unique UNIQUE(Designation);
```

*Il est toujours possible d'insérer **NULL** dans une colonne **UNIQUE** optionnelle.
En revanche, si une ligne contient **NULL** pour une colonne **UNIQUE**, aucune autre ligne de la même table ne pourra contenir la valeur **NULL** pour cette colonne !*

Contrainte CHECK

En plus des contraintes natives, il est possible d'ajouter des contraintes « personnalisées » sur les colonnes.

Le mot-clé **CHECK** permet de définir une condition sur une colonne qui testera toutes les valeurs qui y seront insérées. Si la condition n'est pas respectée, la requête d'insertion est rejetée et le SGBD retourne une erreur.

```
-- VÉRIFIER LA VALEUR D'UN NOMBRE
-- La QteDemandee d'un OFAB doit toujours être positive :
ALTER TABLE OFAB ADD
CONSTRAINT OFAB_QteDemandee_Check CHECK(QteDemandee >= 0);
```

```
-- LIMITER LES VALEURS POSSIBLES D'UNE CHAÎNE DE CARACTÈRES
-- Le Statut d'un OFAB ne peut être que 'EC', 'L' ou 'T' :
ALTER TABLE OFAB ADD
CONSTRAINT OFAB_Statut_Values
CHECK(Statut IN ('EC', 'L', 'T'));
```

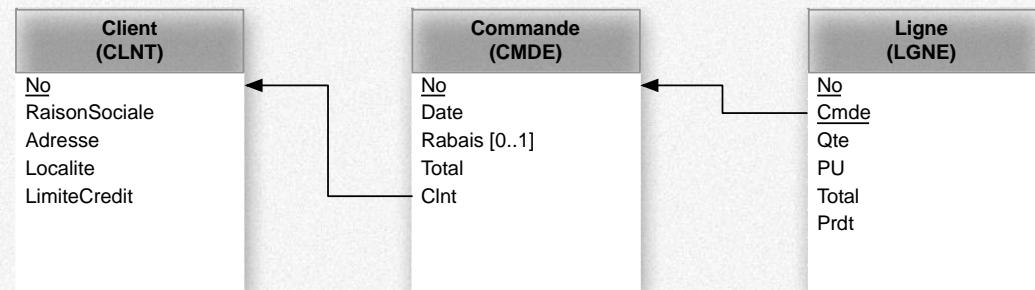
```
-- COMPARER LA VALEUR PAR RAPPORT À UNE AUTRE COLONNE
-- Le montant d'un rabais devrait toujours être égal au
-- pourcentage appliqué sur le total
ALTER TABLE CMDE ADD
CONSTRAINT Commande_Calcul_Rabais
CHECK(MontantRabais = PourcentRabais * Total / 100);
```



Cette contrainte sera évaluée dès que les valeurs de MontantRabais, PourcentRabais ou Total d'une ligne sont ajoutées ou modifiées.

Exercice Production

À vous de mettre à jour votre script de création afin d'y ajouter des contraintes sur les colonnes qui le nécessite.



Client (CLNT)

- La raison sociale devrait être unique.
- La limite de crédit doit toujours être supérieure ou égale à 0.

Commande (CMDE)

- Par défaut, la date de la commande devrait être celle du jour.
- Le total doit toujours être supérieur ou égal à 0.

Ligne (LGNE)

- La quantité, le prix unitaire et le total sont supérieurs ou égaux à 0.
- Le total doit être égal à la quantité multipliée par le prix unitaire.

La fonction GETDATE() fournit la date du jour.

Compléter le fichier

SQLServer_ProductionSchema5_Enonce.sql

Modifier une structure : BD

*Le mot-clé **ALTER** permet la modification d'une structure préalablement créée.*

Nous avons déjà utilisé ce mot-clé pour rajouter les contraintes de clé primaire, de clé étrangère et les contraintes **DEFAULT**, **UNIQUE** et **CHECK**.

Mais **ALTER** permet beaucoup plus de choses.

Il peut s'appliquer aussi bien à une base de données (**ALTER DATABASE**), qu'à une table (**ALTER TABLE**), mais est le plus souvent utilisé pour modifier la structure d'une table.

Sur une base de données, il permet par exemple de modifier le nom de cette dernière :

```
ALTER DATABASE ancien_nom MODIFY NAME = nouveau_nom;  
  
-- Exemple : renommer la base de données Pigeon en Client  
ALTER DATABASE Pigeon MODIFY NAME = Client;
```

Ce genre de requête ne doit pas être exécutée à la légère ; changer le nom d'une base de données après sa création peut avoir des conséquences (très) fâcheuses.

Modifier une structure : Table

Dans un script de création de base de données, on préférera modifier la requête CREATE TABLE.

Les possibilités de modification d'une table sont plus poussées que celles pour une base de données.

Il est ainsi possible d'ajouter une nouvelle colonne à la table :

```
ALTER TABLE nom_de_table ADD  
    nom_de_colonne DATATYPE;  
  
-- Exemple : ajout d'une nouvelle colonne DateAjout  
ALTER TABLE OFAB ADD  
    DateCloture DATETIME2 NOT NULL;
```

Nous avons vu qu'`ALTER` permet aussi de rajouter des contraintes à une table.

Si votre script doit effectuer plusieurs modifications sur une même table, il est possible d'enchaîner les modifications sans effectuer plusieurs `ALTER TABLE`.

```
-- Exemple : Ajout d'une colonne et de plusieurs contraintes  
ALTER TABLE OFAB ADD  
    DateCloture DATETIME2,  
    CONSTRAINT OFAB_Date_Default DEFAULT(GETDATE()),  
    CONSTRAINT OFAB_Statut_Value CHECK(Statut IN ('EC', 'L', 'T')),  
    CONSTRAINT OFAB_QteDemandee_Positive CHECK(QteDemandee >= 0);
```

Modifier une structure : Colonne (1/3)



Lors de la création de la table Meteo, Temperature était de type INT.

La modification d'une colonne revient en fait à modifier une table. Il faudra donc utiliser une instruction ALTER TABLE, suivie d'un ALTER COLUMN.

```
ALTER TABLE nom_de_table  
    ALTER COLUMN nom_de_colonne DATATYPE;
```

```
-- Exemple : Modification d'une colonne en type FLOAT  
ALTER TABLE Meteo ALTER COLUMN Temperature FLOAT;
```

Il faut bien comprendre que la nouvelle déclaration de la colonne écrase complètement l'ancienne déclaration.

C'est-à-dire que si la colonne a été déclarée comme étant NOT NULL lors de la création de la table, il ne faut pas oublier de le rajouter dans l'instruction ALTER COLUMN.

```
CREATE TABLE Maitrise (Niveau BIT NOT NULL);  
-- Après l'instruction suivante, Niveau devient optionnelle  
ALTER TABLE Maitrise ALTER COLUMN Niveau TINYINT;  
-- Il aurait fallu écrire...  
ALTER TABLE Maitrise ALTER COLUMN Niveau TINYINT NOT NULL;
```

Modifier une structure : Colonne (2/3)

Lors de la création de la table STCK, Qte était déjà de type INT, mais pouvait être NULL.



Pour rajouter une contrainte NOT NULL sur une colonne sans modifier son type, il est malgré tout obligatoire d'indiquer le type actuel dans l'instruction :

```
-- La colonne Qte est déjà de type INT  
ALTER TABLE STCK ALTER COLUMN Qte INT NOT NULL;
```

Si une colonne est soumise à une contrainte de type PRIMARY KEY, UNIQUE, DEFAULT ou CHECK, il sera impossible de la modifier en utilisant l'instruction ALTER COLUMN.

```
CREATE TABLE MBRE (Clbr INT NOT NULL, Eqpe INT NOT NULL);  
ALTER TABLE MBRE ADD  
CONSTRAINT MBRE_PK PRIMARY KEY(Clbr, Eqpe);  
  
-- La requête suivante va lever une erreur  
ALTER TABLE MBRE ALTER COLUMN Clbr FLOAT;
```

La seule manière de modifier cette colonne serait de temporairement supprimer la contrainte posant problème, d'exécuter l'instruction ALTER COLUMN, puis de redéfinir la contrainte, en utilisant ALTER TABLE ADD CONSTRAINT.

Modifier une structure : Colonne (3/3)

Si la colonne contient déjà des données, la modification du type de cette colonne n'est possible que si toutes les données peuvent être converties dans le nouveau type.

Supposons la requête suivante :

```
ALTER TABLE Ventes ALTER COLUMN Montant FLOAT NOT NULL;
```

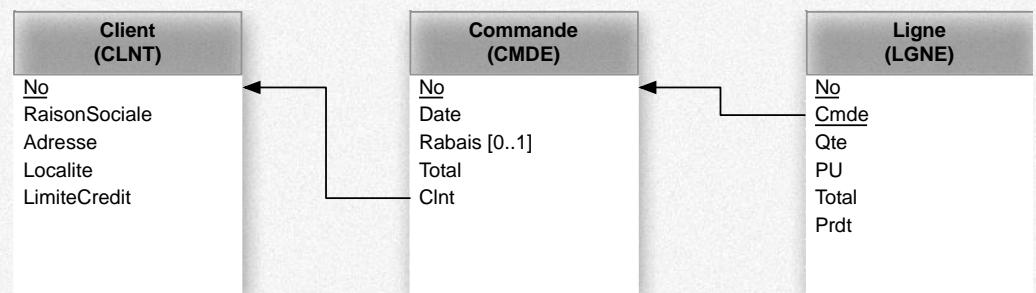
	Montant	[1]
1	100	
2	15000	
3	9800	
4	357000	
5	9900	
6	7650	
7	389	
8	6999	

	montant	[2]
1	Faible	
2	Élevé	
3	Faible	
4	Moyen	
5	Moyen	
6	Élevé	
7	Moyen	
8	Faible	

Dans le cas du tableau [1], la colonne Montant est de type `INT`. Une valeur `INT` peut être convertie en une valeur `FLOAT`. Par conséquent, la requête précédente sera correctement exécutée, puisque toutes les valeurs pourront être converties dans le nouveau type.

En revanche, dans le tableau [2], la colonne Montant est de type `VARCHAR` et contient des valeurs purement textuelles ('Faible', 'Moyen', 'Élevé'). La requête précédente lèvera donc une erreur, puisque toutes les valeurs ne pourront pas du tout être converties dans le nouveau type `FLOAT`.

Exercice Production



Ajout de colonne

Ajoutez une nouvelle colonne DateAjout dans la table CLNT, de type **DATE** et optionnelle.

Modification de colonne

Modifiez la colonne DateAjout de la table CLNT en changeant son type de **DATE** en **DATETIME2**.

Ajout de contrainte

Ajoutez une contrainte telle que par défaut, la DateAjout doit être la date du jour.

Ouvrez un nouvel onglet dans Azure Data Studio pour chacune des deux requêtes à effectuer.

Mettre à jour des données

La mise à jour de données dans une table s'effectue en utilisant le mot-clé UPDATE.

```
UPDATE nom_de_table  
SET colonne1 = newValue1, colonne2 = newValue2 [, ...];
```



Attention ! La syntaxe précédente va s'appliquer à **toutes** les lignes de la table sélectionnée !

```
-- Exemple : modifier le Statut de tous les OFAB en 'L' :  
UPDATE OFAB  
SET Statut = 'L';
```

Pour ne modifier que les valeurs de certaines lignes bien spécifiques, il faut utiliser l'instruction **WHERE** de la même manière que lors d'instructions **SELECT**.



Les clés primaires prennent toutes leur importance avec les instructions UPDATE : il faut être sûr à 100% de pouvoir cibler une ligne de manière précise.

```
-- Exemple : modifier le Statut de l'OFAB n°12 :  
UPDATE OFAB  
SET Statut = 'L'  
WHERE No = 12;
```

Supprimer une ou des ligne(s)

L'instruction DELETE permet de supprimer une ou des lignes d'une table.

```
DELETE FROM nom_de_table  
WHERE <expressions>;
```

La clause WHERE s'utilise comme dans SELECT ou UPDATE :

```
-- Exemple : supprimer tous les OFAB terminés ('T') :  
DELETE FROM OFAB  
WHERE Statut = 'T';
```



DELETE est à utiliser avec parcimonie et en toute connaissance de cause : il est impossible de récupérer les lignes supprimées.

Pour supprimer toutes les lignes d'une table, il suffit de faire une requête DELETE sans indiquer de clause WHERE :

```
-- Exemple : supprimer tous les OFAB :  
DELETE FROM OFAB;
```

Cependant, par souci de performance et de rapidité, il est conseillé d'utiliser l'instruction TRUNCATE TABLE :

```
-- Exemple : supprimer tous les OFAB :  
TRUNCATE TABLE OFAB;
```

Supprimer une colonne

Supprimer une colonne d'une table (et donc toutes les éventuelles données qu'elle contient) revient à modifier la structure d'une table.

Il faut donc utiliser une instruction `ALTER TABLE` associée aux mots-clés `DROP COLUMN` :

```
ALTER TABLE nom_de_table  
DROP COLUMN nom_de_colonne;
```

La suppression d'une colonne d'une table est cependant **impossible** si la colonne est soumise à des contraintes (clé primaire, `CHECK`, `UNIQUE`, ...)

```
-- Reprenons cet exemple de table Membre (MBRE)  
CREATE TABLE MBRE (Clbr INT NOT NULL, Eqpe INT NOT NULL);  
ALTER TABLE MBRE ADD  
CONSTRAINT MBRE_PK PRIMARY KEY (Clbr, Eqpe);
```

```
-- L'instruction suivante va lever une erreur : la colonne  
-- Clbr est utilisée par la contrainte de clé primaire.  
ALTER TABLE MBRE  
DROP COLUMN Clbr;
```



Pour pouvoir exécuter cette requête, il faut d'abord supprimer la contrainte de clé primaire.

Supprimer une contrainte

Supprimer une contrainte n'a **aucune** incidence sur les données qui y sont soumises.

La suppression d'une contrainte revient aussi à modifier une table. Il faut utiliser à nouveau ALTER TABLE, couplé avec DROP CONSTRAINT.

```
ALTER TABLE nom_de_table  
DROP CONSTRAINT nom_de_contrainte;
```

Comme vous le voyez, la commande **DROP CONSTRAINT** nécessite d'indiquer le **nom de la contrainte** à supprimer.

Il est possible de créer des contraintes sans indiquer manuellement le nom (bien que ce SlideDoc ne l'illustre jamais) ; dans ce cas, SQL Server va lui attribuer un nom généré automatiquement.

Il est conseillé de **toujours donner un nom** personnalisé aux contraintes créées, afin d'avoir une maîtrise totale des éléments créés.

```
-- Exemple : suppression de la contrainte de clé primaire :  
ALTER TABLE MBRE  
DROP CONSTRAINT MBRE_PK;
```

Supprimer une table



Un grand pouvoir implique de grandes responsabilités. Utiliser à la légère les instructions de cette page peut avoir des conséquences fâcheuses.

Supprimer une base de données

Pour supprimer une base de données, il faut exécuter la requête dans un contexte autre que celui de la base de données elle-même.

La suppression d'une table s'effectue en utilisant l'instruction

DROP TABLE :

```
DROP TABLE nom_de_table;
```

Supprimer une table efface **toutes les données qui y sont stockées** ainsi que la structure de la table elle-même (colonnes, contraintes, …).

```
-- Exemple : suppression de la table OFAB  
DROP TABLE OFAB;
```

La suppression d'une base de données s'effectue en utilisant l'instruction DROP DATABASE :

```
DROP DATABASE nom_de_table;
```

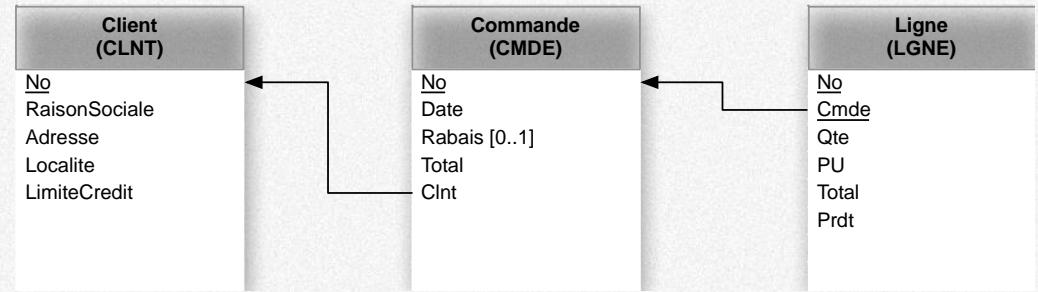
Supprimer une base de données efface **la totalité de son contenu** (données, tables, colonnes, contraintes, …).

```
-- Exemple : suppression de la base de données Assurance  
USE master; -- changement de contexte d'exécution  
DROP DATABASE Production;
```

Exercice Production

Suppression de colonnes

Essayez de supprimer la colonne DateAjout que vous aviez précédemment ajouté à la table CLNT...

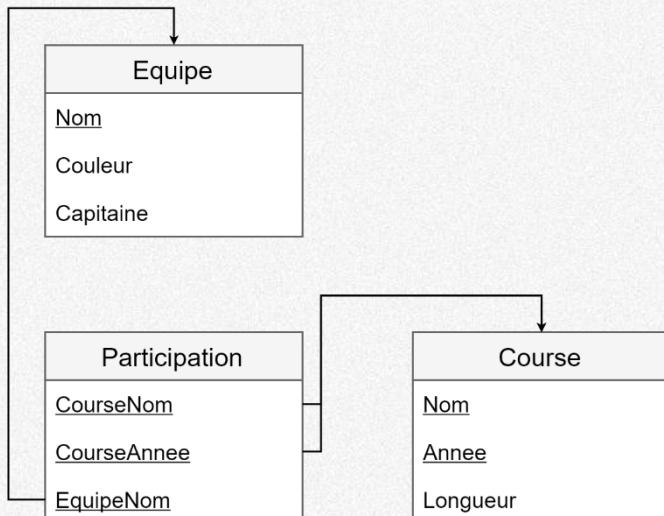


Ouvrez un nouvel onglet dans Azure Data Studio pour ce script.

Clés étrangères composées (1/2)



Ce script fonctionnera, mais
CourseAnnee et CourseNom feront respectivement
référence à Nom et Annee...



Quelques restrictions supplémentaires sont à prendre en compte lors de la création de clé étrangère lorsqu'elle est composée de plusieurs colonnes.

Lorsqu'une clé étrangère fait référence à une clé primaire composée de plusieurs colonnes (exemple ci-dessous), il faut bien faire attention à ce que l'ordre des colonnes lors de la déclaration de la contrainte de clé étrangère corresponde à l'ordre des colonnes de la clé primaire.

-- Création de la table Course :

```
CREATE TABLE Course (
    Nom VARCHAR(30) NOT NULL,
    Annee INTEGER NOT NULL,
    Longueur FLOAT NOT NULL);
ALTER TABLE Course ADD
    CONSTRAINT Course_PK PRIMARY KEY (Nom, Annee)
```

-- Création de la table Participation :

```
CREATE TABLE Participation (
    CourseNom VARCHAR(30) NOT NULL,
    CourseAnnee INTEGER NOT NULL,
    EquipeNom VARCHAR(30) NOT NULL);
ALTER TABLE Participation ADD
    CONSTRAINT Participation_FK
        FOREIGN KEY (CourseAnnee, CourseNom) REFERENCES Course;
```

Clés étrangères composées (2/2)

Avec ce script, CourseNom fera correctement référence à Nom, et CourseAnnee référencera bien Année.

Pour s'en assurer, il n'y a qu'une seule possibilité...

Déclarer **dans le même ordre** les colonnes de la clé primaire et les colonnes de la clé étrangère :

```
-- Création de la table Course :  
CREATE TABLE Course (  
    Nom VARCHAR(30) NOT NULL,  
    Année INTEGER NOT NULL,  
    Longueur FLOAT NOT NULL);  
ALTER TABLE Course ADD  
    CONSTRAINT Course_PK PRIMARY KEY (Nom, Année)  
  
-- Création de la table Participation :  
CREATE TABLE Participation (  
    CourseNom VARCHAR(30) NOT NULL,  
    CourseAnnée INTEGER NOT NULL,  
    EquipeNom VARCHAR(30) NOT NULL);  
ALTER TABLE Participation ADD  
    CONSTRAINT Participation_FK  
        FOREIGN KEY (CourseNom, CourseAnnée) REFERENCES Course;
```

Considérations relatives aux clés étrangères (1/3)



ON DELETE n'est à indiquer que lorsque l'on souhaite modifier le comportement par défaut, qui est la plupart du temps le plus adapté.

Suppression de ligne

Supprimer une ligne dans une table est **interdit par défaut** si la valeur de clé primaire de cette ligne est utilisée comme valeur de clé étrangère dans une autre table.

Lors de la déclaration d'une clé étrangère, il est cependant possible de définir, grâce à **ON DELETE**, et l'un des mots-clés ci-dessous, le comportement à suivre lors d'une tentative de suppression de la clé primaire référencée :

```
-- Syntaxe pour ON DELETE :  
CONSTRAINT nom_contrainte  
FOREIGN KEY (colonne [, ...])  
REFERENCES table  
[ON DELETE {NO ACTION|CASCADE|SET NULL|SET DEFAULT}]
```

NO ACTION

Soulève une erreur et empêche la suppression. **Il s'agit du comportement par défaut.**

CASCADE

Supprime toutes les lignes dont la valeur de clé étrangère correspond à la valeur de la clé primaire de la ligne supprimée.

SET NULL

Toutes les colonnes de clé étrangère référençant la clé primaire supprimée prendront la valeur **NULL** (**seulement si elles ne sont pas NOT NULL**).

SET DEFAULT

Toutes les colonnes de clé étrangère référençant la clé primaire supprimée prendront leur valeur par défaut (**si elle existe... Sinon, la valeur sera NULL, si possible**).

Considérations relatives aux clés étrangères (2/3)

Modification de ligne

De la même manière que pour la suppression, la modification d'une valeur de clé primaire utilisée dans une ligne comme clé étrangère est par défaut impossible.

Lors de la déclaration d'une clé étrangère, il est cependant possible de définir, grâce à `ON UPDATE`, et l'un des mots-clés ci-dessous, le comportement à suivre lors d'une tentative de modification de la clé primaire référencée :

```
-- Syntaxe pour ON UPDATE :  
CONSTRAINT nom_contrainte  
FOREIGN KEY (colonne [, ...])  
REFERENCES table  
[ON UPDATE {NO ACTION|CASCADE|SET NULL|SET DEFAULT}]
```

Les mots-clés `NO ACTION`, `CASCADE`, `SET NULL` et `SET DEFAULT` ont la même signification qu'avec `ON DELETE`.

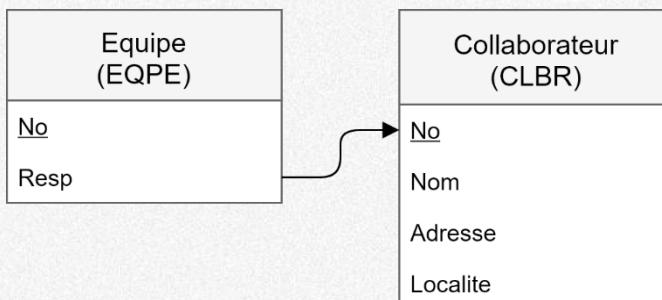
Cette instruction `ON UPDATE` est utile dans les cas où la clé primaire référencée par la clé étrangère est composée de colonne(s) dont les valeurs pourraient être modifiées.

Cela dit, il est généralement préférable de ne jamais modifier une clé primaire.



ON UPDATE n'est à indiquer que lorsque l'on souhaite modifier le comportement par défaut, qui est la plupart du temps le plus adapté.

Considérations relatives aux clés étrangères (3/3)



Suppression des tables

Dans le cas, assez peu courant, où vous souhaiteriez supprimer des tables de la base de données, il est impératif de le faire, là encore, en prenant en compte les références clés étrangères-clés primaires.

En effet, il sera impossible de supprimer une table si sa clé primaire est référencée par une clé étrangère d'une autre table.

Il faudra soit supprimer la table contenant la clé étrangère **avant** la table contenant la clé primaire référencée, soit supprimer la contrainte de clé étrangère de la table qui la contient, puis supprimer la table contenant la clé primaire.

Dans l'exemple ci-contre, pour supprimer la table CLBR, il faudrait donc soit supprimer d'abord la table EQPE ou bien supprimer d'abord la contrainte de clé étrangère présente sur la colonne EQPE.Resp.

-- Supprimer les tables dans le bon ordre :

```
DROP TABLE EQPE;  
DROP TABLE CLBR;
```

-- Supprimer la contrainte puis la table :

```
ALTER TABLE EQPE DROP CONSTRAINT EQPE_Resp_FK;  
DROP TABLE CLBR;
```