Daniel Muller

Introduction au développement d'objets SQL

(Pour Microsoft SQL Server et Sybase ASE)

SOMMAIRE

TYPES DE DONNEES

LES JOINTURES

Création de tables

La table est l'objet le plus important d'une base de données car c'est elle qui va héberger les données. Une table se définit par son nom, ses colonnes et son type. A l'image d'une feuille d'un tableur (Excel, par exemple), les données sont stockées dans des colonnes et on y accède ligne à ligne.

1. Introduction

Un serveur de base de données tel que SQL Server propose quatre types de tables :

- **La table permanente :**

Elle est visible de tous les utilisateurs et sa base maitresse est définie lors de sa création. Sa création est du ressort du DBA et sa suppression ou sa modification ne peut se faire sans son assentiment.

- **La table temporaire locale :**

Généralement créée au sein d'une procédure stockée, elle n'est visible que de son créateur et ne vit que le temps de la session de la procédure ou jusqu'à la déconnexion de son propriétaire. Elle est hébergée dans une base de données dédiée « tempdb » et se caractérise par le symbole « # » qui précède son nom.

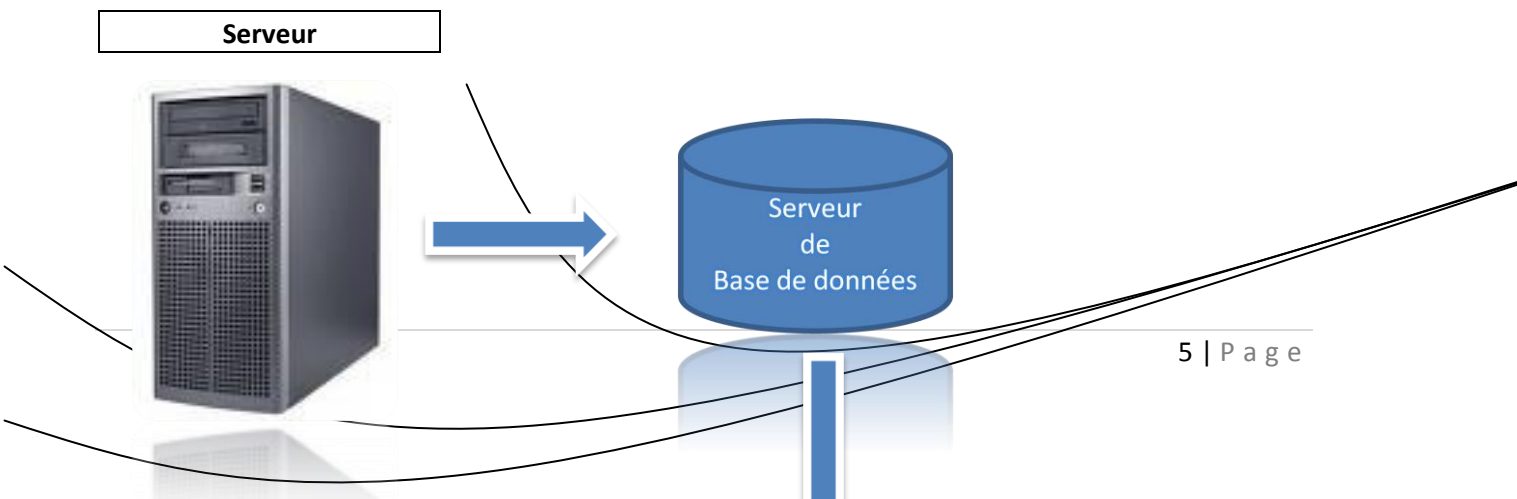
- **La table temporaire globale :**

A l'instar de la table temporaire, elle est le fruit du développeur et sa durée de vie se limite à la durée de connexion de celui-ci. La différence se situe au niveau de la visibilité car, ici, tous les utilisateurs connectés à la base de données peuvent alors accéder à cette table. Pour la différencier d'une table temporaire, son nom est précédé du symbole « ## ».

- **La table variable :**

Alors qu'une table temporaire st systématiquement stockée dans la base « tempdb », la table variable est enregistrée dans la mémoire vive du serveur et donc nettement plus rapide en terme d'accès.

On peut ainsi résumer la structure d'un SGBD par le schéma suivant :



2. Table permanente

La syntaxe de création d'une table permanente est la suivante :

```
CREATE TABLE Base.Proprietaire.Table
(
Colonne_1 TYPE [CONTRAINTE],
Colonne_2 TYPE [CONTRAINTE],
Colonne_X TYPE [CONTRAINTE]
ON 'Fichier de données'
)
```

- **Base :**

Nom de la base de données où sera stockée la table

- **Propriétaire :**

Nom du créateur de la table. Dans la majorité des cas, il s'agit de « dbo » (DataBase Owner).

- **Table :**

Nom de la table qui devra être unique au sein de la base de données.

- **Colonne_X :**

Nom de la colonne au sein de la table. Ce nom doit suivre une norme selon son utilisation (clé primaire, clé étrangère …)

- **TYPE :**

Type de données attribué à la colonne (Cf : Chapitre « Types de données »)

- **CONTRAINTE :**

Contrainte associée à la colonne

- **ON :**

Spécifie dans quel fichier de données la table va être physiquement créée. Cette fonctionnalité est particulièrement utile si la table sera volumineuse et si plusieurs fichiers de données ont été définis au sein de la base de données.

Exemple :

```
CREATE TABLE BaseClient.dbo.DetailAdresse
(
Id_DetailAdresse INT NOT NULL PRIMARY
KEY,
Adresse1 VARCHAR (128) NOT NULL,
Adresse2 VARCHAR (128) NULL,
Ville VARCHAR (64) NOT NULL,
CodePostal INT NOT NULL,
AutresDetails VARCHAR (256) NULL
)
```

La table « DetailAdresse » a pour but d'enregistrer les informations postales d'un client, par exemple. Elle est créé dans la base de données « BaseClient ». Elle contient une colonne de clé primaire, deux colonnes qui permettent de stocker l'adresse (numéro, bâtiment, rue ou avenue …), la ville, le code postal et la possibilité d'ajouter des informations supplémentaires (cedex, boite postale …).

S'il est conseillé de toujours préciser la base source pour éviter de créer la table au mauvais endroit, l'indication du propriétaire, elle, est optionnelle.

3. Table temporaire

L'intérêt principal d'un tel objet est de pouvoir centraliser les données, soit nouvelles, soit issues d'autres tables au sein d'une structure organisée qu'une personne autre que le DBA peut créer, administrer et supprimer sans affecter la base elle-même.

Qu'il s'agisse d'une table temporaire locale ou globale, elle est hébergée dans la base de données « tempdb ». Egalement, sa durée de vie est dépendante de la session de son propriétaire.

Une **table temporaire locale** n'est manipulable et visible que par son créateur. Elle se distingue d'une table permanente par le préfixe « # » à son nom.

Syntaxe :

```
CREATE TABLE #TableTempLocale
(
Colonne_1 TYPE [CONTRAINTE],
Colonne_2 TYPE [CONTRAINTE],
Colonne_X TYPE [CONTRAINTE]
)
```

Pour montrer que la structure d'une table temporaire est identique à celle d'une table permanente, il est tout à fait possible de reprendre la structure de l'exemple précédent :

Exemple :

```
CREATE TABLE #TempDetailAdresse
(
Id_DetailAdresse INT NOT NULL PRIMARY
KEY,
Adresse1 VARCHAR (128) NOT NULL,
Adresse2 VARCHAR (128) NULL,
Ville VARCHAR (64) NOT NULL,
```

```
CodePostal INT NOT NULL,
AutresDetails VARCHAR (256) NULL
)
```

Contrairement à la table temporaire locale, une **table temporaire globale** est visible de tous les utilisateurs tant que son créateur ne l'a pas explicitement supprimé ou qu'il n'a pas clos sa session. Elle se distingue par le préfixe « ## » porté à son nom.

Syntaxe :

```
CREATE TABLE #TableTempGlobale
(
Colonne_1 TYPE [CONTRAINTE],
Colonne_2 TYPE [CONTRAINTE],
Colonne_X TYPE [CONTRAINTE]
)
```

Pour montrer que la structure d'une table temporaire globale est identique à celle d'une table temporaire locale, on peut reprendre l'exemple précédent :

Exemple :

```
CREATE TABLE ##TempDetailAdresse
(
Id_DetailAdresse INT NOT NULL PRIMARY
KEY,
Adresse1 VARCHAR (128) NOT NULL,
Adresse2 VARCHAR (128) NULL,
Ville VARCHAR (64) NOT NULL,
CodePostal INT NOT NULL,
AutresDetails VARCHAR (256) NULL
)
```

A noter que, même si les tables temporaires sont automatiquement supprimées par le moteur SQL en fin de session, l'utilisateur qui a créé une table temporaire sera avisé de la supprimer dès lors qu'il estime ne plus en avoir besoin. Cela libère de l'espace mémoire et allège la base « tempdb »

4. Table variable

Une table variable est, comme son nom l'indique, une table stockée au sein d'une variable. En tant que tel, elle ne peut exister que dans une procédure stockée, une fonction ou un trigger. Sa durée de vie est donc tributaire de ces objets.

Puisqu'elle est considérée comme une variable, elle présente donc l'avantage d'être créée dans un espace de la mémoire vive du serveur de base de données. Le développeur ne développe un tel objet que s'il estime avoir besoin d'un accès rapide aux données enregistrées dans cette table. De plus, la mémoire physique du serveur étant limitée, elle ne doit contenir qu'un nombre restreint de lignes. Si la table devient trop volumineuse, l'espace supplémentaire requis pour les nouvelles informations sera stocké dans la base « tempdb ».

A l'instar de toute variable, une table variable est préfixée par le symbole « @ » et initialisée par la clause « DECLARE ».

Syntaxe :

```
DECLARE @TableVariable TABLE
(
Colonne_1 TYPE [CONTRAINTE],
Colonne_2 TYPE [CONTRAINTE],
Colonne_X TYPE [CONTRAINTE]
)
```

Exemple :

```
DECLARE @VarDetailAdresse
(
Id_DetailAdresse INT NOT NULL PRIMARY
KEY,
Adresse1 VARCHAR (128) NOT NULL,
Adresse2 VARCHAR (128) NULL,
Ville VARCHAR (64) NOT NULL,
CodePostal INT NOT NULL,
AutresDetails VARCHAR (256) NULL
)
```

5. Table temporaire Vs table variable

Si les conditions sont réunies pour la création d'une table variable, il ne faut pas hésiter à y recourir au détriment des tables temporaires. Un tel réflexe est souvent synonyme de gain de performances pour le serveur de bases de données. Il faut cependant le signaler au DBA afin qu'il puisse vérifier que le nombre de tables variables qui peuvent être simultanément appelées ne dépassent pas les limites physiques du serveur.

En effet, la création d'une table temporaire initie un processus qui immobilise une part non négligeable des ressources du SGBD :

- Création de l'objet « table temporaire » dans la base « tempdb »,
- Insertion de données dans la table temporaire,
- Sélection des données avec probablement une ou plusieurs jointures avec la base de données courante,
- Suppression de la table temporaire.

Récupérer des informations d'une table temporaire entraîne un verrouillage de la base « tempdb », verrous que le SGBD sait plus ou moins bien gérer pour éviter des problèmes de contention. Ainsi, cette gestion des tables temporaire est plus la spécialité de SQL Server que de Sybase.

Le comportement d'une table variable est nettement plus efficient car il ne mobilise qu'une part de la mémoire vive du serveur. Par conséquence, pas d'accès disques intempestifs, pas de verrous à administrer, pas de contentions à redouter.

En outre, utiliser une table variable peut éviter l'appel d'un curseur, l'un des éléments les plus contraignants d'une base de données.

L'exemple qui suit va utiliser des tables issues de la base « Northwind » sur SQL Server. L'objectif est de retourner, ligne à ligne, les dénominations d'un vendeur, son pays d'origine, le nombre total de ventes qu'il a passé depuis 2005.

Exemple :

```
DECLARE @Compteur SMALLINT,
                @Max SMALLINT

DECLARE @BestEmployes TABLE
(
Id_BestEmployes INT NOT NULL PRIMARY KEY IDENTITY,
Prenom VARCHAR (128),
NomFamille VARCHAR (128),
Pays VARCHAR (128),
NbreVentes SMALLINT
)
```

```
INSERT INTO @BestEmployes
SELECT E.FirstName,
            E.LastName,
            E.Country,
            Count (O.OrderId)
FROM Employes AS E
INNER JOIN Orders AS O ON E.EmployesId =
O.EmployeeId
WHERE YearDate (O.OrderDate) >= 2005
GROUP BY E.EmployesId,
                  E.LastName
                  E.PostalCode

SELECT @Max = Max (IdBestEmployes)
FROM @BestEmployes

SET @Compteur = 1

WHILE (@Compteur <= @Max)
BEGIN
          SELECT *
          FROM @BestEmployes
          WHERE IdBestEmployes = @Compteur

          SET @Compteur = @Compteur + 1
END
```

Que se passe-t-il exactement ?

- Deux variables sont créées. La première va servir de compteur, la seconde stockera la valeur maximale de l'identifiant de la table variable.
- La table variable « @BestEmployes » est initialisée.
- On insère dans la table variable les informations suivantes :
    - Une clé primaire auto-incrémentale,
    - Le prénom de l'employé,
    - Le nom de famille de l'employé,
    - Le pays de l'employé
    - Le nombre de commandes passés par l'employé.

  Les données prises en comptent démarrent à partir de l'année 2005.
  Deux jointures ont été nécessaires pour récupérer les informations nécessaires.
- La variable « @Max » récupère la valeur maximale de l'identifiant de la table variable.

- La variable « @Compteur » est positionnée à 1.
- Une boucle va lire et afficher, ligne par ligne, les informations stockées dans la table variable.

Résultat :

Identifiant	Prénom	Nom	Pays	Nbre de ventes
1	Nancy	Davolio	USA	97
2	Andrew	Fuller	Irlande	80
3	Jean-Claude	Pinel	France	125
4	Paul	King	Irlande	129
5	Stéphane	Larreur	France	101

6. Sécurité

Il est possible d'accorder des privilèges particuliers ou, au contraire, d'en retirer à un utilisateur qui souhaite accéder à une table.

Grâce à la clause « **GRANT** », l'administrateur ou le développeur, s'il en a les droits, peut offrir à une personne qui peut accéder à la base de données un droit total ou partiel à une table.

Syntaxe :

```
GRANT [ALL] / [METHODE]

ON Nom_Table
TO Utilisateur
```

- **[ALL] :**

Accorde tous les droits associés à une table à l'utilisateur désigné.

- **[METHODE] :**

Désigne le ou les droits auxquels l'utilisateur désigner a accès.

- **Nom_Table :**

Nom de la table désignée

- **Utilisateur :**

Nom ou login de l'utilisateur à qui on associe ces droits.

Le tableau ci-dessous dresse la liste des méthodes d'accès à une table :

METHODE	DESCRIPTION
CREATE	Accorde l'autorisation de créer une table dans la base courante
ALTER	Donne le droit de modifier une table existante dans la base courante
SELECT	Offre le moyen d'afficher les lignes d'une table
INSERT	Permet à l'utilisateur d'ajouter de nouvelles données dans une table
UPDATE	Permet à l'utilisateur de modifier des données existantes
DELETE	Accorde à l'utilisateur le moyen de supprimer une ou plusieurs lignes d'une table

Exemple :

```
USE BaseClient
GO

GRANT INSERT, SELECT, UPDATE
ON DetailsCommande
To PMartin, Cbauer
GO
```

Au niveau de la base de données « BaseClient », les utilisateurs Paul Martin et Chris Bauer, respectivement désignés par les logins « PMartin » et « CBauer », ont le droit d'afficher les données de la table « DetailsCommandes » mais peuvent également y insérer de nouvelles lignes ou de mettre à jour les données existantes. Par contre, ils n'auront pas la possibilité de supprimer des informations sur cette table.

SQL Server offre la possibilité de supprimer des droits à un utilisateur grâce à la clause « **REVOKE** ».

Syntaxe :

```
REVOKE [ALL] / [METHODE]
ON NomTable
TO Utilisateur
```

Exemple :

```
USE BaseClient
GO

REVOKE DELETE
```

```
ON DetailsCommande
To Public
GO
```

Ici, le groupe « Public », qui est celui accordé par défaut à un utilisateur connecté à la base, n'a désormais plus le moyen de supprimer des lignes dans la table « DetailsCommandes ».

Types de données

Lors de la modélisation d'une base de données, une réflexion cruciale doit être portée sur la structure des tables qui la compose. Plus précisément, la définition des colonnes doit être la plus réaliste possible sous peine de connaître des problèmes de dysfonctionnement du SGBD.

En pratique, si la définition d'une colonne est trop petite par rapport aux données qu'elle reçoit, le serveur va non seulement générer des erreurs d'incompatibilité mais également empêcher l'enregistrement de ces informations.

A l'inverse, des colonnes surdimensionnées ne causeront pas de pertes de données mais ces dernières occuperont plus d'espace qu'il ne leur en faut et provoqueront à terme un remplissage rapide des fichiers physiques.

1. Catégories de types de données

Un SGBD tel que SQL Server comprend 6 catégories de données distinctes :

- Données numériques exactes :

- INT (BIGINT, SMALLINT, TINYINT)
- BIT
- DECIMAL
- NUMERIC
- MONEY (SMALLMONEY)

- Données numériques approximatives :

- FLOAT
- REAL

- Données temporelles :

- DATETIME (SMALLDATETIME)

- Données alphanumériques :

- CHAR (NCHAR)
- VARCHAR (NVARCHAR).
- TEXT

- Données binaires :

- BINARY
- VARBINARY
- IMAGE

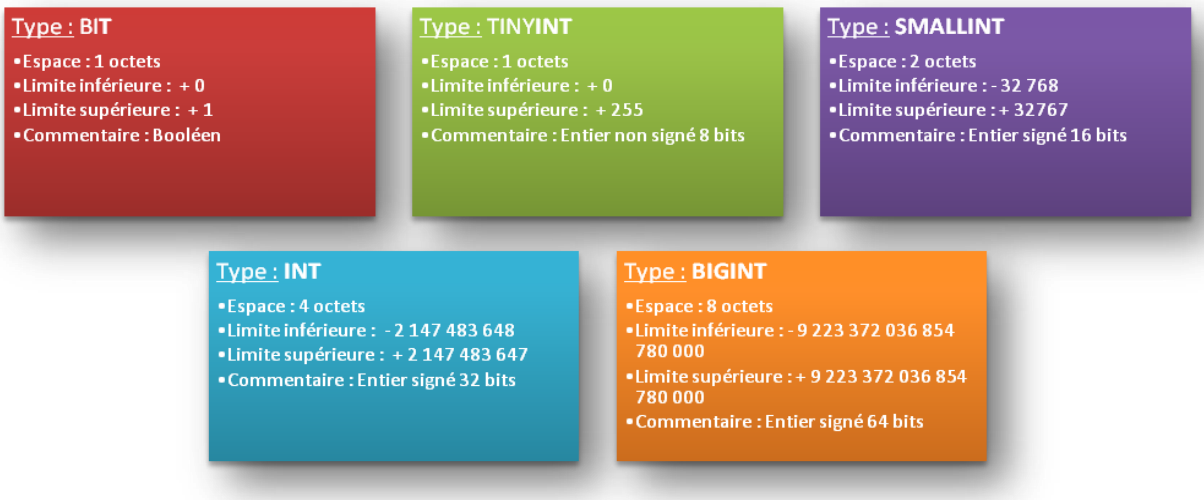
- Données spécifiques :

- TIMESTAMP
- CURSOR

2. Données numériques exactes

Dans cette catégorie peuvent être stockées les valeurs numériques qui peuvent être entiers ou décimaux. Ces nombres sont manipulables par n'importe quel opérateur mathématique.

Le type de données « **INT** » permet le stockage de nombres entiers. Ils peuvent être positifs ou négatifs. Si les limites imposées par ce type ne correspondent pas à votre attente, SQL propose des types de données alternatifs tels que « **SMALLINT** » pour une plage de valeurs plus restreintes ou, au contraire, « **BIGINT** » qui offre un choix plus large. Occasionnellement, le type de données « **TINYINT** » peut être utile pour l'enregistrement de petites valeurs strictement positives. Le type « **BIT** » est consacré à l'enregistrement de valeurs booléennes.



Les types de données **« DECIMAL (P, S)** » et **« NUMERIC (P, S)** » sont consacrés au stockage des nombres décimaux. L'un et l'autre possèdent les mêmes caractéristiques et sont donc interchangeables. Ce qui les caractérise sont les valeurs P et S qui définissent une précision et une échelle fixe.

- Exemple :

La valeur 1865,34 se définit ainsi :

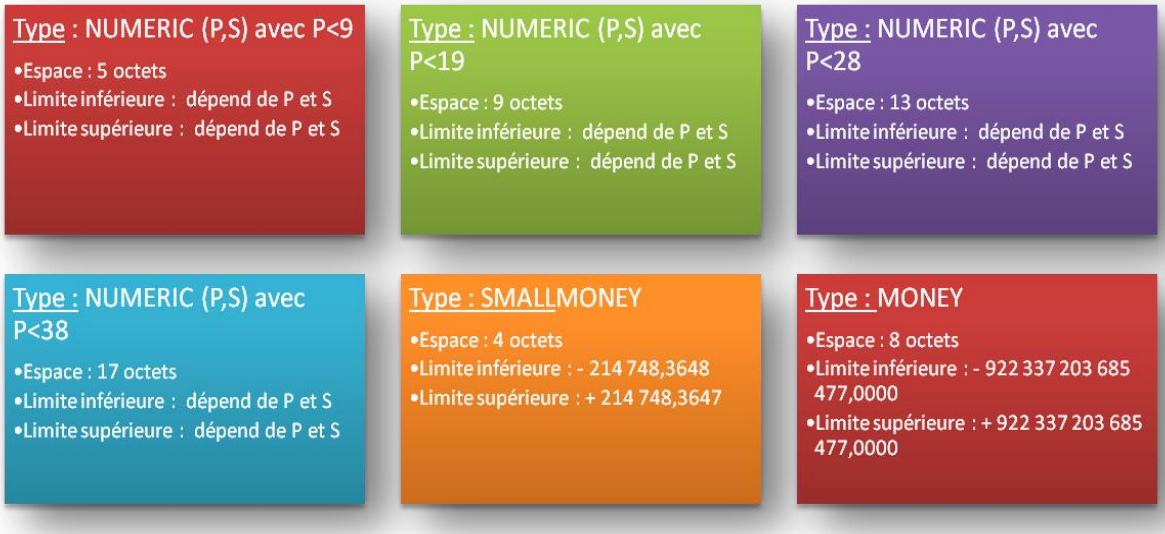
- Précision : 7
- Echelle : 2

La **précision** P correspond au nombre maximal de chiffres situés tant à gauche qu'à droite de la virgule.

L'**échelle** S définit le nombre maximal de chiffres situés à droite de la virgule. Cette valeur doit évidemment être inférieure à la précision P.

L'espace de stockage qu'occupe une donnée de ce type dépend de la valeur de la précision.

Il existe également un type de données exclusivement consacré aux données monétaires : « **MONEY** ». Il se caractérise par une échelle fixée à 4 et par ses limitations en plage de valeurs. La variante « **SMALLMONEY** » peut être utilisée pour des données de taille moins importantes.



3. Données numériques approximatives

Cette catégorie a pour objectif de stocker des nombres décimaux à virgule flottante. La différence avec la catégorie précédente est la notion d'échelle inexistante ici. En effet, dans le cas du type de données « **FLOAT (N)** », seule la notion de précision est utilisée.

De ce fait, un nombre plus grand qu'autorisé par la valeur de la précision sera arrondi avant d'êtres stocké en base.

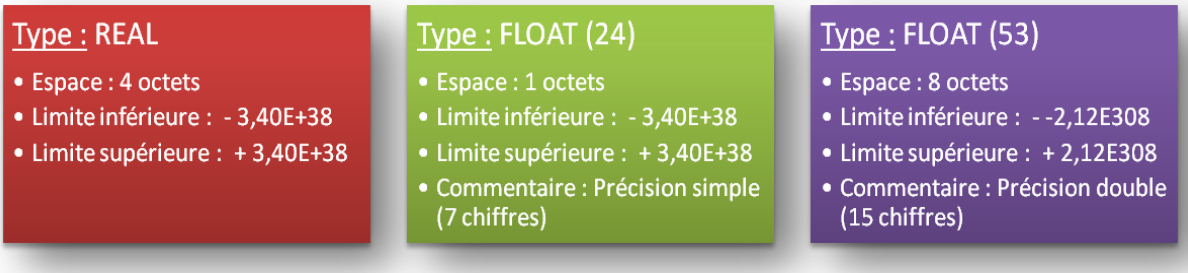
Par exemple, le nombre 198,321523478 sera arrondi s'il doit être pris en charge par une colonne de type FLOAT (8). La valeur que l'on retrouvera en base sera alors : 198,3215.

Il faut noter que la valeur N correspond ici au nombre de bits utilisés pour stocker la **mantisse**. Seules deux valeurs ne sont réellement utilisées : 24 et 53 pour obtenir une précision de 7 ou 15 chiffres.

Il faut rappeler que la mantisse est la différence entre un nombre et sa partie entière :

- Pour le décimal positif 7984,78 : la partie entière est 7984 et la mantisse est 0,78
- Pour le décimal négatif -7984,78 : la partie entière vaut -7985 et la mantisse est 0,22

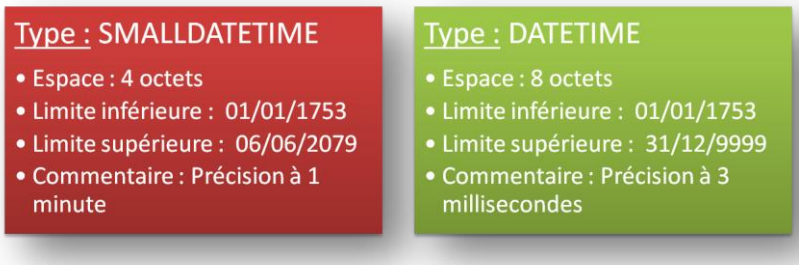
Le type de données « **REAL** » n'est quasiment jamais utilisé. Il correspond en fait à FLOAT (24).



4. Données temporelles

Le stockage des informations d'ordre chronologique est une réelle difficulté dans le monde des bases de données. Les formats de dates au sein des applications clientes peuvent se présenter sous divers formats, aussi la base doit-elle enregistrer l'information sans perte et résister à toute manipulation ultérieure.

SQL Server ou Sybase proposent deux types de données qui offrent des niveaux de précision différents. Si l'on recherche à enregistrer des données fiables au-dessous de la seconde, on va alors privilégier le type « **DATETIME** ». Si une précision à la minute près suffit, on optera alors pour le type « **SMALLDATETIME** ». Ce choix a son importance car les données enregistrées en « DATETIME » occupent un espace double par rapport au « SMALLDATTIME ».



5. Données alphanumériques

Lorsque l'on doit stocker des informations alphanumériques, trois questions se posent :

- Tout d'abord, connaît-on la taille exacte des données à insérer ? Cela est possible si elles font l'objet d'une normalisation (Code INSEE, numéro de département etc. …). On utilisera donc le type « **CHAR (x)** » dont la longueur est fixée à x caractères.

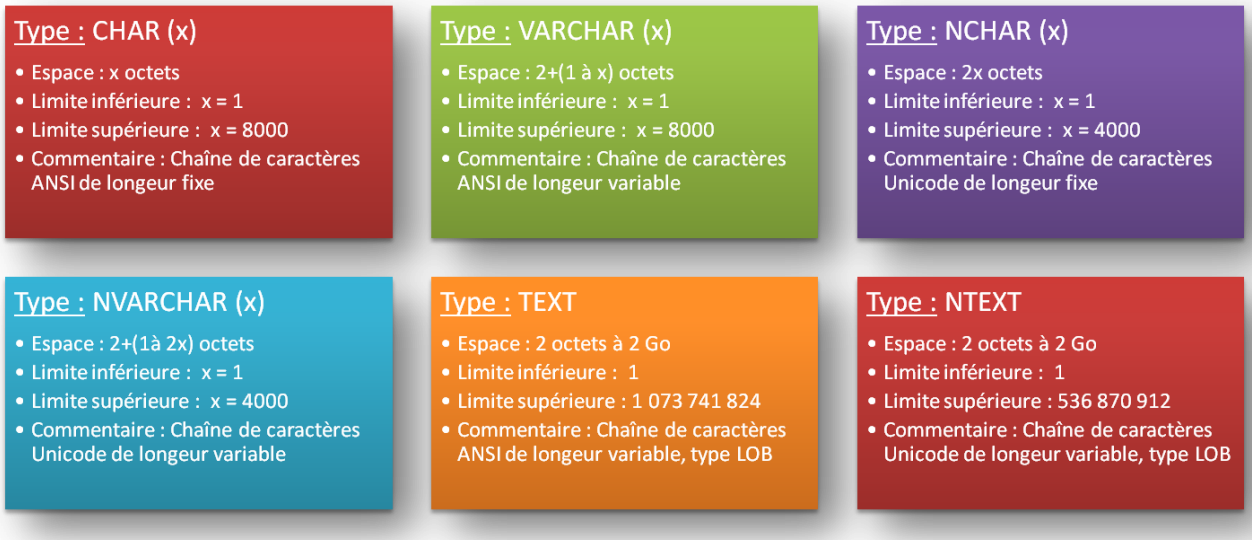
Si, au contraire, les données peuvent être de tailles diverses, c'est le type « **VARCHAR (x)** » qu'il faudra choisir car l'espace occupé sur le disque dépendra réellement du nombre de caractères qui les composent.

- Ensuite, les données peuvent-elles provenir ou, à l'inverse, sont-elles destinées à une application utilisant un classement différent de votre base de données ?

  Si c'est effectivement le cas, SQL Server propose de stocker les informations sous le format Unicode. Les informations sont ainsi « transportables » d'un serveur à un autre quelle que soit la langue utilisée sur celui-ci. Les types adéquats sont alors « **NCHAR (x)** » et « **NVARCHAR (x)** ». L'inconvénient de standard est qu'un caractère Unicode occupe deux fois plus d'espace qu'un caractère ANSI.

- Enfin, si les informations sont-elles réellement volumineuses, dépassant de ce fait les limites imposées par les types précédents ?

  Il est possible alors de les stocker grâce à un type de données particulier : « **TEXT** » qui existe également en ANSI et en Unicode avec les restrictions qui en découlent. Ce type fait partie de la famille des « **LOB** » ou 'Large OBject'.



A noter que SQL Server 2005 a abandonné le type TEXT au profit du type **VARCHAR (Max)** lorsque l'information à stocker dépasse la limite des 8000 octets.

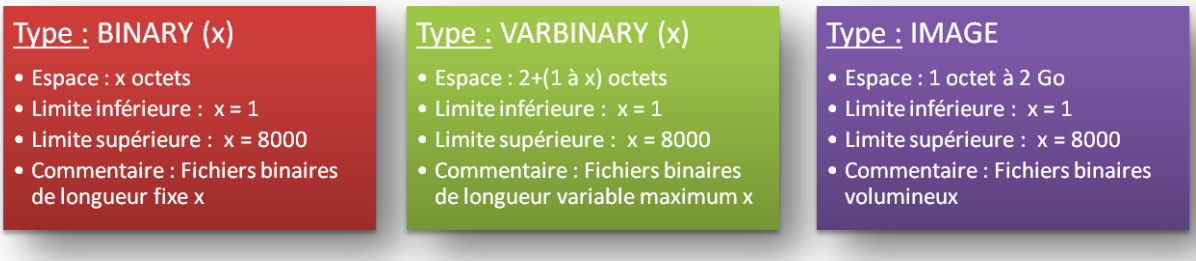
6. Données binaires

En fait de données binaires, cette catégorie offre la possibilité de stocker des fichiers, qu'il s'agisse de photos, de document ou de fichiers compressés. Ceux-ci ne sont-ils pas en fait des paquets de bits ?

Pour des fichiers d'un poids inférieur à 8000 octets dont la taille est plus ou moins constante, le type « **BINARY (x)** » est recommandé. La valeur x représente le nombre d'octets que doit occuper le fichier.

Si, au contraire, les fichiers à stocker sont de tailles très variables, c'est le type « **VARBINARY (x)** » qu'il faudra utiliser.

Quant aux fichiers volumineux, tant qu'ils n'atteignent pas la limite de 2Go, SQL Server propose le type « **IMAGE** ».



A noter que, pour une question de standardisation, SQL Server 2005 a abandonné le type IMAGE pour le type **VARBINARY (Max)** lorsque l'information à stocker dépasse la limite des 8000 octets.

7. Données hors catégories

Il existe enfin des types de données que l'on ne peut classer dans aucune des catégories précédentes, soit parce qu'ils ne peuvent être utilisés en tant que type de colonnes, soit parce que leur utilisation est dédiée à des besoins spécifiques.

« **CURSOR** » est le plus courant de ces types hors catégories car il offre de grands services aux développeurs. En effet, ce type de données n'a d'existence qu'au sein d'une procédure stockée et non dans une table.

En résumé, le rôle d'un curseur est de parcourir les colonnes d'une table ou plusieurs tables déclarées dans une clause « **SELECT** » et ce, ligne à ligne. Chaque donnée est stockée dans une variable locale puis manipulé. Le résultat de ce traitement est renvoyé en sortie, puis le curseur passe à la ligne suivante.

Exemple de curseur :

```
DECLARE @Nom VARCHAR (64),
        @Prenom VARCHAR (64)

DECLARE curseur_auteurs CURSOR FOR
SELECT auteur_prenom,
       auteur_nom
FROM auteurs
```

```
OPEN curseur_auteurs

FETCH curseur_auteurs INTO @Prenom, @Nom

WHILE (@@FETCH_STATUS = 0)
BEGIN
        PRINT @Prenom + ' ' + @Nom
        FETCH curseur_auteurs INTO @Prenom, @Nom
END

CLOSE curseur_auteurs

DEALLOCATE curseur_auteurs
```

Dans cet exemple, Le curseur « Curseur_Auteurs » va parcourir ligne à ligne les colonnes 'auteur_nom' et 'auteur_prénom' de la table 't_auteurs'.

Tant qu'il n'aura pas atteint la dernière ligne de cette table, le curseur va alors exécuter le code au sein de la clause « BEGIN … END ».

Lorsque la dernière ligne sera lue, le moteur va finalement libérer la mémoire allouée au curseur.

Les jointures sont un moyen en Transact-SQL pour récupérer des informations provenant de plusieurs objets, qu'il s'agisse de tables, tables temporaires ou vues en utilisant, si possible, leur clefs primaires et étrangères.

Il existe différents types de jointures, chacune répondant à des besoins précis :

- *Les jointures internes,*
- *Les jointures externes,*
- *Les jointures multiples,*
- *L'auto-jointure.*

1. Constitution des tables d'exemple

Si la base Northwind fournit un certain nombre d'objets intéressants, de nouvelles tables sont nécessaires pour illustrer au mieux les différents cas de jointures.

Imaginons le cas d'une société possédant plusieurs départements. Pour gérer son personnel, elle va utiliser une base de données et les informations seront stockées dans les tables suivantes :

- Departement,
- Emploi,
- Personnel,
- Client,
- Hierarchie.

▪ Création des tables

```
CREATE TABLE Departement
(
Id_departement INT NOT NULL CONSTRAINT
pk_id_departement PRIMARY KEY,
Nom_departement VARCHAR (128) NOT NULL
)
```

```
CREATE TABLE Emploi
(
Id_emploi INT NOT NULL CONSTRAINT pk_id_emploi
PRIMARY KEY,
Description_emploi VARCHAR (128) NOT NULL
)
```

```
CREATE TABLE Personnel
(
Id_personnel INT NOT NULL CONSTRAINT pk_id_personnel PRIMARY KEY,
Nom_personnel VARCHAR (64) NOT NULL,
Departement_fk INT NULL CONSTRAINT fk_departement_fk REFERENCES Departement
(Id_departement),
Emploi_fk INT NOT NULL CONSTRAINT fk_emploi_fk REFERENCES Emploi (Id_emploi),
Salaire DEC (7,2) NOT NULL
)
```

```
CREATE TABLE Client
(
Id_client INT NOT NULL CONSTRAINT pk_id_client PRIMARY KEY,
Nom_client VARCHAR (128) NOT NULL,
Client_ref CHAR (1) NOT NULL CONSTRAINT chk_client_ref CHECK (Client_Ref IN ('O','N'))
)
```

```
CREATE TABLE Hierarchie
(
Id_Hierarchie INT NOT NULL CONSTRAINT pk_id_hierarchie PRIMARY KEY,
Nom VARCHAR (64) NOT NULL,
Prenom VARCHAR (64) NOT NULL,
Responsable_Fk INT NULL CONSTRAINT responsable_fk REFERENCES
Hierarchie (Id_Hierarchie),
)
```

- **Insertion des données**

```
INSERT INTO Departement VALUES (100, 'Ingénierie')
INSERT INTO Departement VALUES (200, 'Production')
INSERT INTO Departement VALUES (300, 'Comptabilité')
INSERT INTO Departement VALUES (400, 'Direction')
INSERT INTO Departement VALUES (500, 'Partenariat')
```

```
INSERT INTO Emploi VALUES (10, 'Informaticien')
INSERT INTO Emploi VALUES (20, 'Technicien')
INSERT INTO Emploi VALUES (30, 'Expert Comptable')
INSERT INTO Emploi VALUES (40, 'Directeur')
INSERT INTO Emploi VALUES (50, 'Manager')
```

```
INSERT INTO Personnel VALUES (1, 'Paul Kaiser', 100, 10, 2700.00)
INSERT INTO Personnel VALUES (2, 'Angela Mitchell', 300, 30, 4200.00)
INSERT INTO Personnel VALUES (3, 'Alain Maurison', 200, 20, 1480.00)
INSERT INTO Personnel VALUES (4, 'Isabelle Tcheky', 400, 40, 5700.00)
INSERT INTO Personnel VALUES (5, 'Gilbert Menon', 400, 50, 5300.00)
INSERT INTO Personnel VALUES (6, 'Jean Picard', NULL, 20, 1510.00)
```

```
INSERT INTO Client VALUES (1, 'SOMATRA', 'O')
INSERT INTO Client VALUES (2, 'USINOR', 'N')
INSERT INTO Client VALUES (3, 'PLASTEX', 'O')
INSERT INTO Client VALUES (4, 'GONDRAND & FILS', 'O')
INSERT INTO Client VALUES (5, 'CASPER ET FRERES', 'N')
```

```
INSERT INTO Hierarchie VALUES (1, ‘Thenan’, ' Gérard',
NULL)
INSERT INTO Hierarchie VALUES (2, ‘Michel’, 'Louise',1)
INSERT INTO Hierarchie VALUES (3, ‘Keil’, ‘Paul’, 2)
INSERT INTO Hierarchie VALUES (4, ‘Pelon’, ‘Géraldine’, 2)
INSERT INTO Hierarchie VALUES (5, ‘Matserki’, ‘Luc’, 4)
```

2. Rappel : Ecriture d'une clause « SELECT »

Une clause 'SELECT' s'écrit selon la syntaxe suivante :

```
SELECT "Liste des colonnes"
FROM Table
WHERE "Condition"
GROUP BY "Liste des colonnes"
HAVING "Condition"
ORDER BY "Liste des colonnes" ("DESC » | « ASC »)
```

3. Jointure interne

Une jointure interne lie deux tables par une condition d'égalité et ne retourne que les informations conformes à cette condition. Cette égalité porte de préférence sur les liens établis entre les deux tables, généralement la clé primaire table et la clé étrangère de la seconde table.

Syntaxe SQL 89 :

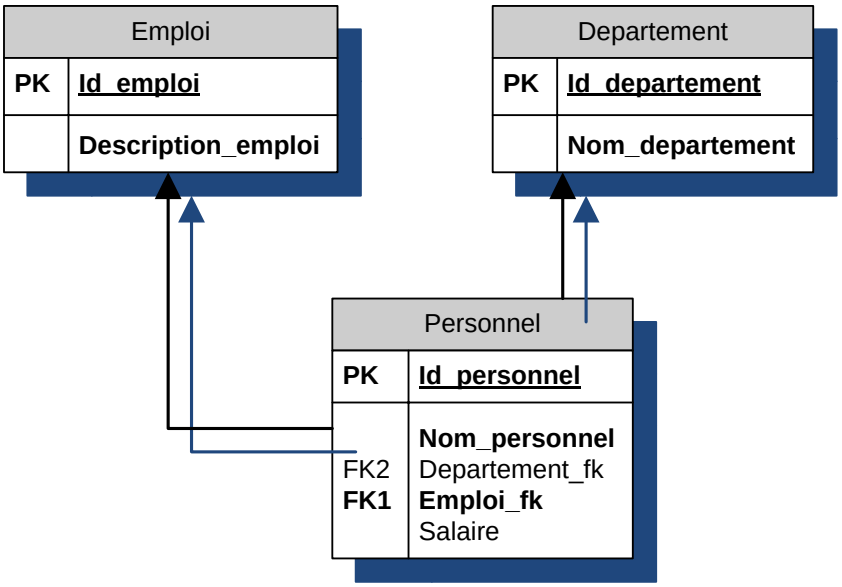
```
SELECT "Liste des colonnes"
FROM  Table1 AS T1,
          Table AS T2
WHERE T1. « Clé primaire » = T2.« Clé étrangère »
```

Syntaxe SQL92 :

```
SELECT "Liste des colonnes"
FROM  Table1 AS T1
 INNER JOIN Table2 AS T2 ON T1. « Clé primaire » = T2.« Clé étrangère »
WHERE « Condition »
```

Exemple :

On souhaite afficher la liste des salariés ainsi que leur déplacement attitré. Pour cela, les tables « Personnel » et « Departement » seront utilisées.



Le schéma ci-dessus montre que les tables sont liées par la relation suivante :

- La clé primaire « Id_departement » de la table « Departement » est associée à la clé étrangère « Departement_fk » de la table « Personnel ».
- La clé primaire « Id_emploi » de la table « Emploi » est associée à la clé étrangère « Emploi_fk » de la table « Personnel »

Selon la définition, la table de gauche est celle qui contient la clé primaire, soit « Departement » dans notre exemple. La table de droite sera celle qui porte la clé étrangère, soit « Personnel ».

La requête s'écrit alors sous les formes suivantes :

Syntaxe SQL 89 :

```
SELECT  P.Nom_Personnel,
        D.Nom_departement
FROM    Departement AS D,
        Personnel AS P
WHERE   D.Id_departement = P.Departement_fk
```

Syntaxe SQL92 :

```
SELECT  P.Nom_Personnel,
        D.Nom_departement
FROM    Departement AS D
INNER JOIN Personnel AS P ON D.Id_departement = P.Departement_fk
```

La syntaxe à préférer est SQL-92. L'ancienne normalisation est montrée ici à titre d'exemple mais, désormais, toutes les requêtes respecteront la plus récente.

Résultat :

Nom_personnel	Nom_departement
Paul Kaiser	Ingenieurie
Angela Mitchell	Comptabilité
Alain Maurison	Production
Isabelle Tcheky	Direction
Gilbert Menon	Direction

Remarque :

La clause « **INNER JOIN** » ne prend en compte que les informations répondant à la condition de jointure. Aussi, les données qui n'ont pas de liens n'apparaissent pas dans le résultat final. Ainsi :

- Jean Picard n'a pas de département attitré
- Le département « Partenariat » ne compte aucun salarié.

Sur SQL Server, il est possible de simplifier l'écriture de la requête en supprimant le mot « INNER » car la jointure interne est le fonctionnement par défaut de ce SGBD.

4. Jointures externes

Une jointure externe permet de retourner des lignes correspondant à la condition de jointure mais aussi toutes celles de la première table, de la seconde table ou des deux qui ne vérifient pas cette condition.

Pour cela, trois expressions existent :

- Pour une jointure externe gauche : « **LEFT OUTER JOIN** »
- Pour une jointure externe droite : « **RIGHT OUTER JOIN** »
- Pour une jointure externe complète : « **FULL OUTER JOIN** »

- **Jointure externe gauche :**

L'expression « **LEFT OUTER JOIN** » retourne toutes les lignes répondant à la condition de jointure (à l'image de la jointure interne) mais aussi toutes celles de la table gauche qui n'ont pas de correspondance avec les lignes de la table de droite.

Syntaxe :

```
SELECT « Liste des colonnes »
FROM    Table1 AS T1
        LEFT OUTER JOIN Table2 AS T2 ON T1.« Clé primaire » = T2.« Clé étrangère »
```

Exemple :

```
SELECT P.Nom_personnel,
        D.Nom_departement
FROM    Departement AS D
        LEFT OUTER JOIN Personnel AS P ON D.Id_departement = P.departement_fk
```

Résultat :

Nom_personnel	Nom_departement
Paul Kaiser	Ingenieurie
Alain Maurison	Production
Angela Mitchell	Comptabilité
Isabelle Tcheky	Direction
Gilbert Menon	Direction
NULL	Partenariat

Remarque :

Il est à noter que la table de gauche est bien « Departement » et la table de droite, « Personnel ». Ce sont donc toutes les lignes de la table « Departement » qui sont retournées, y compris celles sans correspondance avec la table « Personnel ».

- **Jointure externe droite :**

L'expression « **RIGHT OUTER JOIN** » retourne toutes les lignes correspondant à la condition de jointure, puis affiche les lignes de la table de droite qui n'ont pas de correspondance avec celles de la table de gauche.

Syntaxe :

```
SELECT « Liste des colonnes »
FROM    Table1 AS T1
        RIGHT OUTER JOIN Table2 AS T2 ON T1.« Clé primaire » = T2.« Clé étrangère »
```

Exemple :

```
SELECT P.Nom_personnel,
        D.Nom_departement
FROM    Departement AS D
        RIGHT OUTER JOIN Personnel AS P ON D.Id_departement = P.departement_fk
```

Résultat :

Nom_personnel	Nom_departement
Paul Kaiser	Ingenieurie
Alain Maurison	Production
Angela Mitchell	Comptabilité
Isabelle Tcheky	Direction
Gilbert Menon	Direction
Jean Picard	NULL

Remarque :

Il est à noter que la table de gauche est bien « Departement » et la table de droite, « Personnel ». Ce sont donc toutes les lignes de la table « Personnel » qui sont retournées, y compris celles sans correspondance avec la table « Departement ».

- **Jointure externe complète :**

L'expression « **FULL OUTER JOIN** » effectue sa tâche en 3 étapes :

- Elle retourne en premier lieu les lignes correspondant à la condition de jointure.
- En second lieu, elle affiche les lignes de la table gauche sans correspondance avec la table de droite.
- En dernier lieu, elle affiche les données de la table de droite sans liens avec la table gauche.

En résumé :

"FULL OUTER JOIN" = "INNER JOIN" + "LEFT OUTER JOIN" + "RIGHT OUTER JOIN"

Syntaxe :

```
SELECT « Liste des colonnes »
FROM    Table1 AS T1
        FULL OUTER JOIN Table2 AS T2 ON T1.« Clé primaire » = T2.« Clé étrangère »
```

Exemple :

```
SELECT P.Nom_personnel,
        D.Nom_departement
FROM    Departement AS D
        FULL OUTER JOIN Personnel AS P ON D.Id_departement = P.departement_fk
```

Résultat :

Nom_personnel	Nom_departement
Paul Kaiser	Ingenieurie
Alain Maurison	Production
Angela Mitchell	Comptabilité
Isabelle Tcheky	Direction
Gilbert Menon	Direction
NULL	Partenariat
Jean Picard	NULL

5. Jointures multiples

Les exemples cités jusqu'à présent offraient des combinaisons de jointures entre deux tables. Il est cependant possible de construire une requête basée sur des jointures entre tables, voire plus encore.

Les conditions de jointure s'écriront simplement les unes à la suite des autres.

La syntaxe ci-dessous montre une requête de jointure simple entre une table T1 qui est liée à une table T2 mais aussi à une table T3 par l'intermédiaire de deux clés étrangères distinctes.

Syntaxe SQL-89 :

```
SELECT « Liste des colonnes »
FROM    Table1 AS T1,
            Table2 AS T2,
            Table3 AS T3
WHERE T2. « Clé primaire » = T1. « Clé étrangère »
AND       T3. « Clé primaire » = T1. « Clé étrangère »
```

Syntaxe SQL-92 :

```
SELECT « Liste des colonnes »
FROM    Table1 AS T1,
            INNER JOIN Table2 AS T2. « Clé primaire » =  T1. « Clé étrangère »
            INNER JOIN Table3 AS T3. « Clé primaire » = T1. « Clé étrangère »
```

Exemple :

```
SELECT P.Nom_personnel,
            D.Nom_departement,
            E.Description_emploi,
           P.Salaire
FROM    Personnel AS P
            INNER JOIN Departement AS D ON D.Id_departement = P.Departement_fk
            INNER JOIN Emploi AS E ON E.Id_emploi = P.Emploi_fk
```

Résultat :

Nom_personnel	Nom_departement	Description_emploi	Salaire
Paul Kaiser	Ingénierie	Informaticien	27000.00
Alain Maurison	Production	Expert Comptable	42000.00
Angela Mitchell	Comptabilité	Technicien	1480.00
Isabelle Tcheky	Direction	Directeur	5700.00
Gilbert Menon	Direction	Manager	5300.00

6. Auto-jointure

Parfois, certaines tables contiennent une colonne de clé étrangère qui appelle la clé primaire de cette même table. Pour visualiser les données, il est donc nécessaire de créer une liaison entre ces deux clefs. La distinction entre les deux tables sera possible grâce aux alias.

Syntaxe SQL-92 :

```
SELECT T1.« Liste des colonnes de la table T1 »,
        T2.« Liste des colonnes de la table T2 »,
FROM    Table1 AS T1,
        INNER JOIN Table1 AS T1. « Clé primaire » =  T2. « Clé étrangère »
```

Exemple :

Prenons le cas de la table « Hierarchie ». Elle dresse une liste de personne et la colonne « Responsable_fk » permet d'identifier à quel supérieur chacune d'elle est liée.

Par exemple, le directeur n'a pas de supérieur attitré, d'où la valeur « NULL » qui lui est assignée dans cette colonne.

Dans un cadre de normalisation, les commandes du langage T-SQL sont classées en quatre familles distinctes. Ainsi les ordres qui permettent de manipuler les données sont dans une catégorie différente des ordres qui sont chargés de vérifier la bonne exécution d'une requête au sein d'une transaction.

1. Catégorie : Définition des objets

- **Data Definition Langage (DDL) :**

Cette famille regroupe les commandes du langage T-SQL qui permettent de créer des objets au sein du SGBD, qu'il s'agisse de base de données, de tables, de vues, d'index ou encore de contraintes.

Les commandes sont les suivantes :

- **CREATE**,
  Pour créer un nouvel élément dans la base de données

- **ALTER**,

Pour modifier un objet existant au sein de la base de données

- **DROP**,

Pour supprimer un objet présent dans la base de données.

2. Catégorie : Manipulation des données

- **Data Manipulation Langage (DML) :**

Cette famille regroupe les commandes T-SQL chargées de traiter les données au sein des objets de la base. Il s'agit généralement d'afficher ou de manipuler les informations renfermées dans des tables ou dans des vues.

Les commandes sont les suivantes :

- **SELECT,**

  Pour extraire et afficher un résultat issu de l'interrogation d'un objet de la base.

- **INSERT,**

  Pour ajouter de nouvelles données dans une table.

- **ALTER,**

  Pour modifier une donnée existante dans un objet de la base.

- **DELETE,**

  Pour supprimer des données présentes au sein d'un objet de la base.

3. Catégorie : Sécurité des accès aux objets

- **Data Control Langage (DCL) :**

Cette famille regroupe les commandes T-SQL qui permettent d'accorder ou non les droits d'accès aux objets de la base de données.

Les commandes sont les suivantes :

- **GRANT,**

  Pour autoriser un utilisateur à accéder à un objet de la base de données

- **REVOKE,**

  Pour supprimer l'accès d'un utilisateur à un objet de la base de données

4. Catégorie : Contrôle des transactions

- **Transaction Control Langage (TCL) :**

Les commandes classées dans cette famille ont pour tâche de contrôler l'exécution d'une transaction et annuler les modifications introduites par elle si besoin est.

Les commandes sont les suivantes :

- **BEGIN TRAN[SACTION],**

  Pour indiquer au moteur SQL le point de départ d'une transaction qui sera marquée spécifiquement dans le journal des transactions.

- **END TRAN[SACTION],**

  Pour signaler au moteur SQL le point d'arrêt de la transaction.

- **COMMIT TRAN[SACTION],**

  Pour valider les modifications apportées par la requête qui suit la commande BEGIN TRAN.

- **ROLLBACK TRAN[SACTION],**

  Pour annuler toutes les modifications introduites depuis le début de la transaction.