

EXPLORONS, EXPÉRIMENTONS ET RENFORÇONS NOS CONNAISSANCES

GESTIONNAIRE DE BASE DE DONNÉES – PARTIE 1

DOMINIQUE LIEFFRIG

www.clicparclic.eu

dlieffrig@hotmail.com

EAFC
CENTRE ARDENNE

Table des matières

I NOTIONS THÉORIQUES CONCERNANT LES BASES DE DONNÉES 1

1 APPROCHE DES BASES DE DONNÉES 2
1.1 Préambules 2
1.2 Définition 2
1.3 Illustration 2
1.4 Systèmes de gestion de bases de données 3
1.4.1 Liens entre les données 3
1.4.2 Cohérence des données 3
1.4.3 Souplesse d'accès aux données 3
1.4.4 Sécurité 3
1.4.5 Partage des données 4
1.4.6 Indépendance des données 4
1.4.7 Performances 4
1.4.8 Administration et contrôle 4
2 LE MODÈLE RELATIONNEL 5
2.1 Niveaux de représentation 5
2.1.1 Niveau conceptuel 5
2.1.2 Niveau logique 5
2.1.2.1 Modèle hiérarchique 5
2.1.2.2 Modèle réseau ou navigationnel 6
2.1.2.3 Modèle relationnel 6
2.1.3 Niveau physique 7
2.1.4 Niveau externe 7
2.2 Le modèle relationnel 7
2.2.1 Notion de relation 7
2.2.2 Schéma relationnel 7
2.2.3 Quelques définitions 8
2.2.4 Algèbre relationnelle 9
2.2.4.1 Somme (ou union) 9
2.2.4.2 Différence 10
2.2.4.3 Produit cartésien 10
2.2.4.4 Intersection 11
2.2.4.5 Jointure 12
2.2.4.6 Équijointure 13
2.2.4.7 Jointure naturelle 14
3 COMPLÉMENTS THÉORIQUES CONCERNANT LES NOTIONS DE TABLES, DE COLONNES, DE CONTRAINTES, ... 15
3.1 Table, ligne et colonne 15
3.2 Rôles d'une colonne 15
3.3 Structure et contenu d'une base de données 17
3.4 Conventions de représentation 18
3.5 Un exemple de base de données relationnelle 19
3.6 Modification des données et contraintes d'intégrité 20
3.6.1 Les identifiants 20
3.6.2 Les contraintes référentielles 20
3.6.3 Les colonnes obligatoires 21

II LE LANGAGE SQL 22

1 PRÉSENTATION DE LA BASE DE DONNÉES EXEMPLE 23
1.1 Présentation 23
1.2 Analyse des besoins 23
1.3 Présentation des tables 23
2 GÉNÉRALITÉS CONCERNANT LE LANGAGE SQL 26
2.1 Introduction 26
2.2 Les objets 26
2.2.1 Types d'objets 26

2.2.1.1 La gestion des données ... 26
2.2.1.2 Le stockage physique ... 27
2.2.1.3 Le stockage d'instructions ... 27
2.2.1.4 La gestion des utilisateurs ... 27
2.2.2 Dénomination ... 27
2.3 Les catégories d'instructions ... 27
2.3.1 Le DDL / LDD (Data Definition Language / Langage de définition des données) ... 27
2.3.2 Le DML / LMD (Data Manipulation Language / Langage de manipulation des données) ... 27
2.3.3 Le TCL (Transaction Control Language / Langage de contrôle des transactions) ... 27
3 DESCRIPTION ET MANIPULATION DES OBJETS ... 28
3.1 L'instruction CREATE TABLE ... 28
3.1.1 Présentation ... 28
3.1.2 Types de données ... 28
3.1.2.1 Types de données numériques exacts ... 28
3.1.2.2 Types de données numériques à virgule flottante ... 29
3.1.2.3 Types de données monétaires ... 29
3.1.2.4 Types de données « date et heure » ... 29
3.1.2.5 Types de données « caractère » ... 29
3.1.2.6 Types de données « binaire » ... 30
3.1.2.7 Types de données spécialisés ... 30
3.1.3 Exemples de création d'une table ... 31
3.1.4 Spécification d'une clé primaire ... 32
3.1.5 Spécification d'une clé étrangère ... 32
3.1.6 Contrainte de valeur avec la clause CHECK ... 34
3.1.7 Attribution d'une valeur par défaut ... 35
3.2 Modification d'une structure de table avec ALTER TABLE ... 36
3.2.1 Introduction ... 36
3.2.2 Changement d'une caractéristique de colonne NOT NULL en NULL ... 36
3.2.3 Changement d'une caractéristique de colonne NULL en NOT NULL ... 37
3.2.4 Augmentation d'une largeur de colonne ... 38
3.2.5 Réduction d'une largeur de colonne ... 38
3.3 Utilisation d'une clé primaire ... 39
3.3.1 Introduction ... 39
3.3.2 Déclaration d'une clé primaire lors de la création d'une table ... 39
3.3.3 Déclaration d'une clé primaire sur une table existante ... 40
3.4 Clé étrangère et intégrité référentielle ... 40
3.4.1 Introduction ... 40
3.4.2 Déclaration d'une clé étrangère lors de la création d'une table ... 40
3.4.3 Déclaration d'une clé étrangère sur une table existante ... 41
3.4.4 Vérification des relations entre clés primaires et étrangères ... 42
3.4.5 Activation ou désactivation des contraintes d'intégrité ... 43
3.4.6 Suppression d'une clé primaire ... 43
3.4.7 Suppression d'une clé étrangère ... 44
3.5 Déclaration d'une contrainte d'unicité ... 44
3.5.1 Présentation ... 44
3.5.2 Différences entre clés primaires et contraintes d'unicité ... 45
3.6 Index de table ... 45
3.6.1 Présentation ... 45
3.6.2 Création d'un index ... 46
3.6.3 Suppression d'un index ... 46
3.6.4 Réserves sur la création d'un index UNIQUE ... 46
3.7 Visualisation des contraintes ... 46
3.7.1 Utilisation de DESCRIBE ... 46
3.7.2 Utilisation de vues systèmes afin de visualiser les contraintes ... 47
3.8 Limitation de valeurs avec la contrainte CHECK ... 48
3.8.1 Présentation ... 48
3.8.2 Contraintes CHECK multiples sur une colonne ... 50
3.8.3 Référence à d'autres colonnes dans une contrainte CHECK ... 50
3.9 Limitations de modification d'une structure de table ... 52
3.9.1 Introduction ... 52
3.9.2 Modification d'une clé primaire ... 52

3.10 Script de création de l'exemple de base de données 52
3.11 Résumé 55
4 MODIFICATIONS DES DONNÉES AVEC SQL 57
4.1 Langage de manipulation des données 57
4.2 Recherche et modification des données 57
4.3 Grammaire SQL 57
4.4 Syntaxe de l'instruction INSERT 58
4.5 Ajout de lignes avec INSERT 59
4.5.1 Introduction 59
4.5.2 Cohérence entre le type de la valeur insérée et celui de la colonne 59
4.5.3 Utilisation de pseudo-colonnes et de fonctions en tant que valeurs 60
4.5.4 Spécification des colonnes dans une instruction INSERT 60
4.6 Utilisation d'une sous-requête : syntaxe de l'instruction INSERT incluant une sous-requête 61
4.7 Modification de données avec UPDATE 62
4.7.1 Syntaxe de l'instruction UPDATE 62
4.7.2 Modification de valeurs dans plusieurs colonnes 62
4.8 Permutation de valeurs avec UPDATE 62
4.9 Utilisation d'une sous-requête avec UPDATE 63
4.10 Suppression de données avec DELETE 65
4.10.1 Syntaxe de l'instruction DELETE 65
4.10.2 Suppression de toutes les lignes d'une table avec TRUNCATE TABLE 65
4.11 Accès simultanés au système d'information 66
4.12 Attribution d'une valeur NULL 66
4.12.1 Valeur NULL avec INSERT 66
4.12.2 Valeur NULL avec UPDATE 66
4.12.3 Valeur par défaut et valeur NULL 67
4.13 Transaction : unité logique de travail 67
4.13.1 Présentation 67
4.13.2 Validation d'une transaction avec COMMIT 68
4.13.3 Annulation d'une transaction avec ROLLBACK TRANSACTION 68
4.13.4 Points d'enregistrement 69
4.14 Résumé 70
5 SÉLECTION DE DONNÉES EN UTILISANT LE LANGAGE SQL 71
5.1 Syntaxe de l'instruction SELECT 71
5.1.1 Introduction 71
5.1.2 Instruction SELECT simple 71
5.1.3 Liste de sélection 71
5.1.4 Résultats retournés par une instruction SELECT 71
5.1.5 Utilisation d'expressions dans une instruction SELECT 72
5.1.6 Opérateurs arithmétiques et logiques 72
5.1.7 Opérateurs pour chaînes 72
5.1.8 Fonctions intégrées 74
5.2 Spécification d'un critère dans la clause WHERE 74
5.2.1 Présentation 74
5.2.2 Combinaison d'expressions avec AND et OR 74
5.3 Tri de données avec la clause ORDER BY 75
5.4 Comptage des lignes d'une table 76
5.5 Recherche des lignes contenant une valeur NULL 76
5.6 Spécification d'un critère avec l'opérateur LIKE 77
5.7 Requête avec l'opérateur BETWEEN 79
5.8 Requête avec l'opérateur IN 80
5.9 Sélection de colonnes avec un alias 81
5.10 Utilisation d'une sous-requête 81
5.10.1 Introduction 81
5.10.2 Création d'une nouvelle table avec l'instruction SELECT 82
5.11 Résumé 83
6 UTILISATION DES FONCTIONS INTÉGRÉES EXISTANT EN SQL 84
6.1 Manipulation de chaînes 84
6.1.1 Recherche de la longueur d'une chaîne 84

6.1.2 Extraction d'une portion de chaîne 84
6.1.3 Recherche d'une portion de chaîne 85
6.1.4 Remplacement d'une portion de chaîne 87
6.1.5 Amputation d'une chaîne 88
6.1.6 Remplissage d'une chaîne 90
6.1.6.1 Fonction LPAD (disponible avec Oracle, MySQL) 90
6.1.6.2 Fonction RPAD (disponible avec Oracle, MySQL) 92
6.1.6.3 Fonction REPLICATE (disponible avec SQL Server) 92
6.1.7 Modification de la casse dans une chaîne 92
6.1.8 Récupération d'une chaîne avec la fonction DECODE (uniquement disponible avec Oracle) 93
6.1.9 Conversion d'un caractère en sa valeur numérique ASCII 94
6.2 Manipulation de données 94
6.2.1 Types de données DATETIME et SMALLDATETIME 94
6.2.2 Stockage des informations d'heures avec les types DATETIME, SMALLDATETIME 95
6.2.3 Obtention de la date et de l'heure courantes avec GETDATE() 95
6.2.4 Formats de date 96
6.3 Déclaration d'une colonne numérique 96
6.3.1 Présentation 96
6.3.2 Utilisation de fonctions intégrées pour les statistiques 97
6.3.3 Nombres arrondis et tronqués explicitement 97
6.3.3.1 Fonction ROUND 98
6.3.3.2 Fonction FLOOR 98
6.3.3.3 Fonction CEILING 98
6.3.4 Recherche des valeurs extrêmes 98
6.3.5 Déterminer si une valeur est NULL 99
6.4 Résumé 100
7 REQUÊTES SQL AVANCÉES 102
7.1 Fonctions de groupe 102
7.1.1 Introduction 102
7.1.2 Fonction COUNT 102
7.1.3 Fonctions MAX et MIN 102
7.1.4 Fonctions AVG et SUM 103
7.1.5 Combinaison de colonnes et de fonctions de groupe 103
7.2 Recherche de lignes distinctes 103
7.3 Groupement de lignes – Les clauses GROUP BY et HAVING 104
7.4 Gestion d'informations hiérarchiques 107
7.4.1 Introduction 107
7.4.2 L'opérateur EXISTS 109
7.5 Opérations de jointure 110
7.5.1 Présentation 110
7.5.2 Jointure simple de deux tables 110
7.5.2.1 Etude de la syntaxe sur base d'exemples 110
7.5.2.2 Colonnes ambiguës 112
7.5.3 Produit cartésien de deux tables 112
7.5.4 Requêtes sur des données récursives 112
7.5.5 Jointure externe 114
7.5.6 Comment interpréter une requête complexe ? 114
7.6 Instruction SELECT avec opérateur d'ensemble 114
7.6.1 Opérateur INTERSECT 115
7.6.2 Opérateur UNION 116
7.6.3 Opérateur EXCEPT 117
7.7 Création d'une table à partir d'une autre table 118
7.8 Résumé 118

III MISE EN PRATIQUE 119

1 PRÉSENTATION 120
2 CONTEXTE 120
3 CRÉATION DU MODÈLE CONCEPTUEL DES DONNÉES (MCD) 120
3.1 Recueil des informations 120
3.2 Constitution du dictionnaire des données 120

3.3 Graphe des dépendances fonctionnelles 121
3.4 Construction du modèle conceptuel des données (MCD) 123
4 CRÉATION DU MODÈLE LOGIQUE DES DONNÉES (MLD) 124
4.1 Présentation, méthodologie et exemple 124
4.2 Les formes normales 125
4.2.1 Principe et utilité de la normalisation 125
4.2.2 Les formes normales 125
4.2.2.1 La première forme normale (1 FN) 125
4.2.2.2 La deuxième forme normale (2 FN) 125
4.2.2.3 La troisième forme normale (3 FN) 125
5 CRÉATION DU MODÈLE PHYSIQUE DES DONNÉES 126
6 IMPLÉMENTATION DE LA BASE DE DONNÉES 127
6.1 Présentation de la base de données créée – Niveau physique de la modélisation des données 127
6.2 Outils utilisés 129
6.2.1 Installation du moteur du système de gestion de base(s) de données 129
6.2.2 Installation de l'interface de gestion 136
6.3 Créer la base de données et les tables 137
6.3.1 Créer la base de données et les tables via le concepteur de tables de SQL Server Management Studio (SSMS) 137
6.3.1.1 Créer le noyau de la base de données 137
6.3.1.2 Créer la structure des tables 138
6.3.1.3 Créer des relations entre les tables 139
6.3.2 Créer la base de données et les tables en utilisant le langage SQL 142
6.3.2.1 Créer le noyau de la base de données 142
6.3.2.2 Créer la structure des tables et les relations 142
6.3.3 Modifier la définition (structure) d'une table 143
6.3.3.1 Modifier la structure d'une table via le concepteur de tables de SQL Server Management Studio 143
6.3.3.2 Modifier la structure d'une table via le langage SQL 144
6.3.4 Supprimer une table 145
6.3.4.1 Supprimer une table via les fonctions graphiques de SQL Server Management Studio 145
6.3.4.2 Supprimer une table en utilisant le langage SQL 145
7 AJOUTER, METTRE À JOUR, SUPPRIMER DES ENREGISTREMENTS DANS LES TABLES 146
7.1 Ajouter, mettre à jour, supprimer des enregistrements en utilisant les fonctions graphiques de SQL Server Management Studio 146
7.1.1 Ajouter des enregistrements dans une table 146
7.1.2 Modifier des données dans une table 146
7.1.3 Supprimer des données dans une table 147
7.2 Ajouter, mettre à jour, supprimer des enregistrements en utilisant le langage SQL 147
7.2.1 La commande INSERT 148
7.2.2 La commande UPDATE 149
7.2.3 La commande DELETE 149
8 CONSULTER LES DONNÉES 150
8.1 Consulter les données en utilisant l'interface graphique de SSMS 150
8.1.1 Afficher le contenu d'une table 150
8.1.2 Extraire des données d'une table avec définition de critères 150
8.1.3 Extraire des données provenant de différentes tables avec définition de critères 153
8.1.4 Enregistrer une requête en tant que vue 154
8.2 Consulter les données en utilisant le langage SQL 156
8.2.1 La commande SELECT 156
8.2.1.1 Créer des requêtes portant sur une seule table 156
8.2.1.2 Créer des requêtes avec opération de regroupement 157
8.2.1.3 Créer des requêtes avec sous-requêtes 158
8.2.1.4 Créer des requêtes de type « jointure » 159
8.2.2 Créer, modifier, utiliser et supprimer une vue 159

I Notions théoriques concernant les bases de données

1 Approche des bases de données

1.1 Préambules

Depuis de nombreuses années le concept de bases de données est devenu relativement familier au grand public. Une base de données peut être vue de façon grossière comme une collection de données gérées par un ordinateur et accessibles par plusieurs utilisateurs en même temps.
Dans les années 60, se sont développées des applications qui permettaient de gérer de grandes quantités de données. Ces premières applications géraient les données dans des fichiers. À cette époque la gestion des informations se faisait via des langages tels que : PL1, Pascal, ou le Cobol qui a toujours été le langage de prédilection pour la gestion des fichiers. De nombreux efforts ont été faits afin d'améliorer les supports physiques de stockage des données ainsi que les méthodes d'accès à l'information.
Cependant, au sein des entreprises, les données sont devenues de plus en plus complexes et de plus en plus interdépendantes, c'est ainsi que le concept de « fichier » a été remplacé par celui de « base de données ».

1.2 Définition

« Une base de données est un ensemble structuré d'informations portant sur un sujet déterminé. Cet ensemble de données est stocké de manière cohérente, exhaustive et non redondante dans la mémoire de masse d'un ordinateur ».
On peut également ajouter qu'une base de données doit permettre de satisfaire plusieurs utilisateurs simultanément.

1.3 Illustration

Soit une petite société qui vend différents produits.

Au sein de cette entreprise, on peut distinguer quatre fonctions principales :

- répertorier les produits à vendre;
- répertorier les clients;
- répertorier les factures qui résultent des ventes;
- répertorier les distributeurs des produits.

Toutes les informations sont stockées dans une armoire qui comporte quatre tiroirs :
1° tiroir: produits
2° tiroir: clients
3° tiroir: factures
4° tiroir: distributeurs

Dans le tiroir « Produits » par exemple, on aura un ensemble de fiches. Chacune d'elles permet de caractériser un produit.
Sur une fiche on pourrait avoir:

- n° du produit
- libellé (ou désignation)
- prix d'achat
- prix de vente
- quantité en stock

Dans les trois autres tiroirs, on aura des fiches du même type permettant de caractériser soit les clients, soit les factures, soit les distributeurs. Il faut également ajouter que sur les fiches du tiroir « Factures » par exemple, des informations permettent de créer des liens avec le tiroir « Clients » et le tiroir « Produits ».

Analogie :

- L'armoire représente la base de données.
- Les différents tiroirs correspondent à différentes tables.
- Les différentes fiches d'un tiroir correspondent aux différents enregistrements du fichier.
- Les différentes caractéristiques présentes sur une fiche correspondent aux différents champs d'un enregistrement.

1.4 Systèmes de gestion de bases de données

Un système de gestion de bases de données (SGBD) est un logiciel permettant à l'utilisateur d'agir sur la base de données.
Plus concrètement, l'utilisateur pourra : - définir des données;
- mettre à jour les informations;
- consulter la base de données.

De plus, un système de gestion de bases de données doit permettre de réaliser des tâches en relation avec les notions suivantes :

- liens entre les données;
- cohérence des données;
- souplesse d'accès aux données;
- sécurité;
- partage des données;
- indépendance des données;
- performances;
- administration et contrôle.

1.4.1 Liens entre les données

La gestion des fichiers traditionnels ne permet pas de créer aisément des liens complexes entre les données. Un SGBD quant à lui permettra de définir une multitude de liens entre les données. On sera aidé dans cette tâche par l'élaboration d'un modèle des données.

1.4.2 Cohérence des données

Un SGBD doit permettre à l'utilisateur un maintien de la cohérence d'une base de données.
Pour ce faire, il devra permettre de définir:

- certaines propriétés relatives aux données;
- des règles permettant de limiter certaines manipulations à des groupes d'utilisateurs autorisés.

1.4.3 Souplesse d'accès aux données

Un SGBD doit permettre à l'utilisateur d'accéder facilement à n'importe quelle donnée de la base.
Les langages de requêtes pourront être utilisés de façon interactive par des utilisateurs pour consulter une base de données ou faire des modifications.

1.4.4 Sécurité

La sécurité des données constitue un point essentiel dans la gestion des informations d'une entreprise.
Un SGBD performant devra donc permettre de protéger les données contre les agressions « physiques » et « humaines ».
Par « agressions physiques », on veut parler des pannes d'un périphérique de stockage. On peut d'autre part envisager les erreurs logicielles.

D'un point de vue humain, on peut avoir des erreurs de manipulation volontaires ou involontaires de la part de l'utilisateur.
Le meilleur moyen de protéger les données vis à vis de tels problèmes consiste à prévoir des sauvegardes et des points de reprise permettant de redémarrer le système et de le remettre dans un état de fonctionnement.

1.4.5 Partage des données

Le partage des données entre plusieurs applications constitue une raison d'être des bases de données. Il est donc fondamental que le SGBD :

- permette un tel partage;
- offre les moyens de contrôler le partage des données (détecter les conflits d'accès...).

1.4.6 Indépendance des données

Un SGBD doit permettre d'écrire des applications sans se soucier de la structuration physique des données et des méthodes d'accès associées. De cette façon, le système d'information pourra évoluer sans remettre en cause les applications déjà écrites.

On distingue : - l'indépendance physique.
- l'indépendance logique.

a) L'indépendance physique

L'indépendance physique consiste à rendre l'utilisation des données indépendante de leur organisation physique (support de stockage et méthodes d'accès). Par exemple, la modification de l'organisation physique ne doit pas entraîner la réécriture des programmes permettant d'accéder à ces données.

b) L'indépendance logique

Ce niveau d'indépendance a pour but de permettre d'enrichir une base de données existante pour prendre en compte de nouvelles structures sans pour autant remettre en cause celles qui existent déjà. L'indépendance logique permet donc de faire face à de nouveaux besoins, ce qui est indispensable, si l'on considère qu'une base de données est un modèle du monde réel et que ce dernier, de même que les besoins des utilisateurs, changent au cours du temps.

1.4.7 Performances

Un SGBD doit permettre :

- de gérer un volume important de données;
- aux utilisateurs d'accéder rapidement aux informations.

NB : D'autres caractéristiques sont requises, elles ont été exposées précédemment.

1.4.8 Administration et contrôle

Une base de données sera contrôlée par un administrateur.

Ses fonctions seront les suivantes :

- décider de l'information contenue dans la base;
- choisir les structures physiques et les stratégies d'accès;
- définir les autorisations accordées aux utilisateurs;
- déterminer les points de reprise et les sauvegardes;
- optimiser l'organisation physique afin d'augmenter les performances globales du système;
- ...

2 Le modèle relationnel

2.1 Niveaux de représentation

Quand on envisage la représentation des données contenues dans une base de données, on peut distinguer différents niveaux :

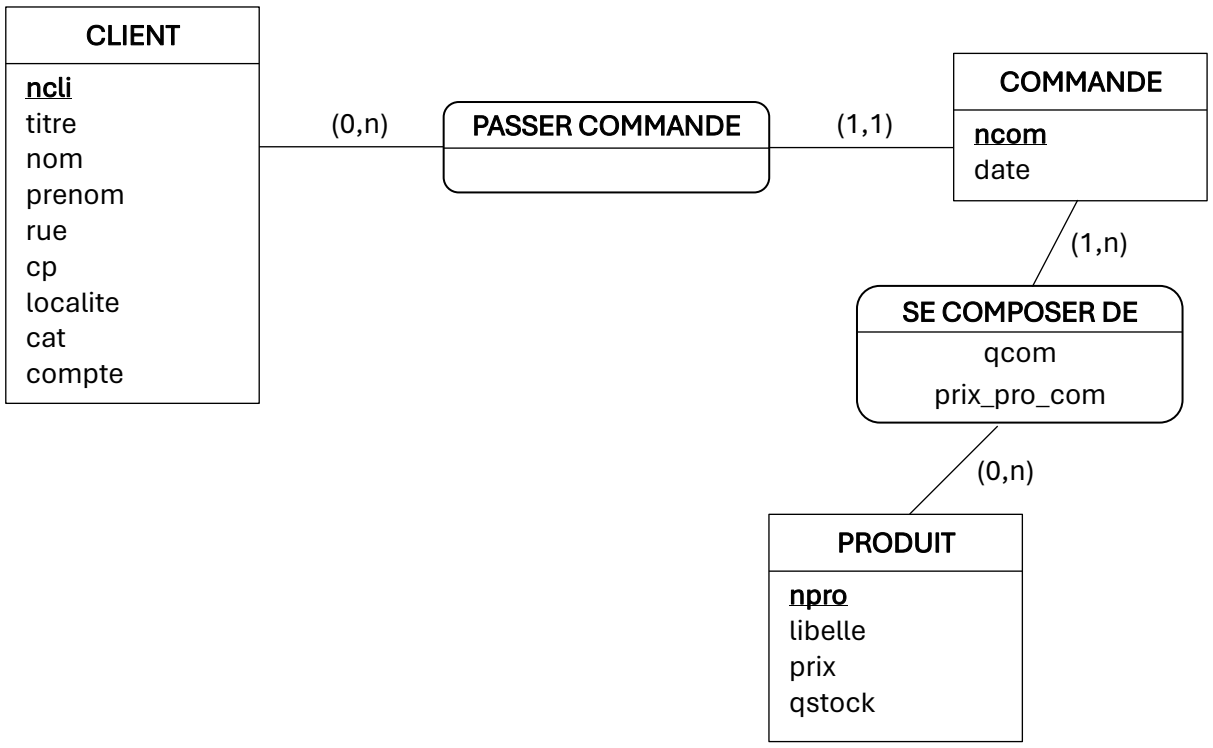
- niveau conceptuel;
- niveau logique;
- niveau physique.

On peut également envisager une représentation de niveau externe.

2.1.1 Niveau conceptuel

Au niveau conceptuel, l'objectif est de modéliser les entités du monde réel et cela sans se préoccuper des moyens physiques qui seront utilisés ultérieurement.

Exemple de modèle conceptuel des données (MCD) :

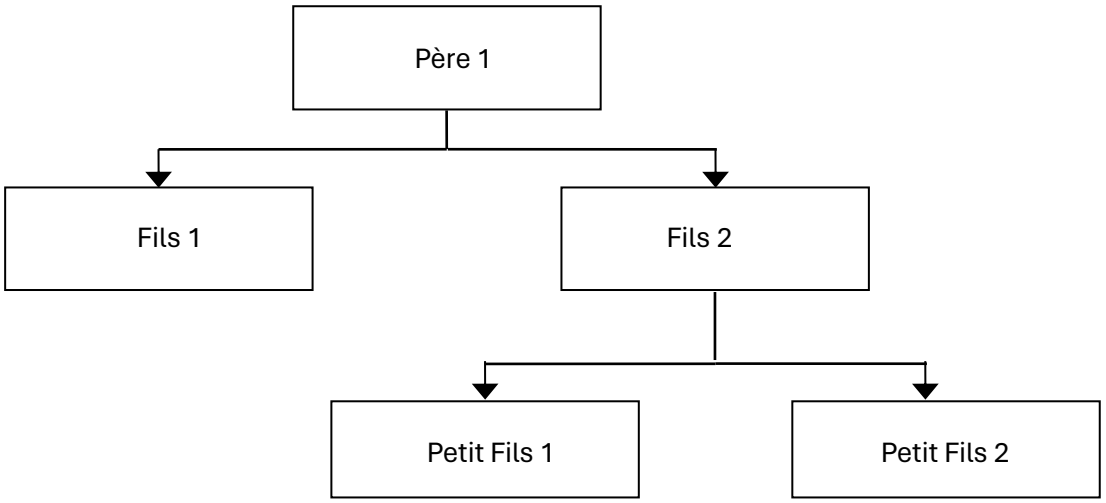


2.1.2 Niveau logique

Le modèle logique des données a pour but de représenter l'univers des données décrit dans le modèle conceptuel des données (MCD) mais en tenant compte du type de base de données choisi : hiérarchique, navigationnel, relationnel, objet...

2.1.2.1 Modèle hiérarchique

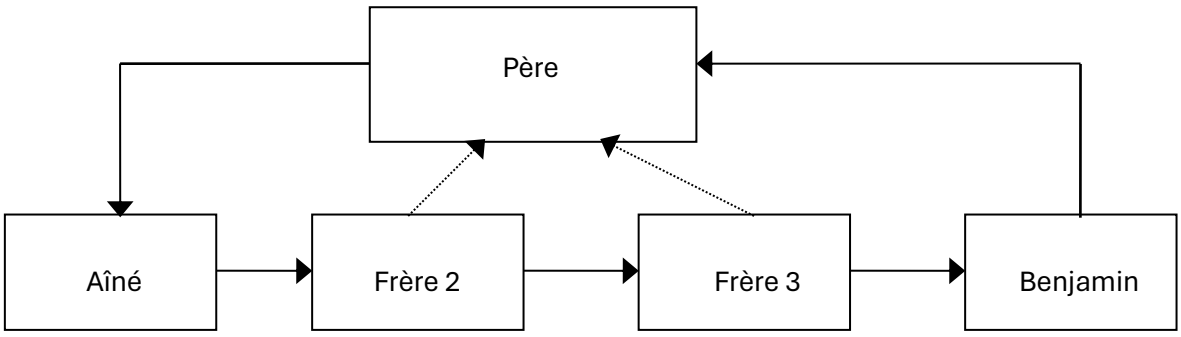
Ce modèle de SGBD est le premier apparu. Les données sont stockées sous forme d'un arbre généalogique où un fils n'a qu'un père et chaque élément est père et/ou fils.



Pour accéder à un petit fils, il faut d'abord transiter par la bonne occurrence de père, puis la bonne occurrence de fils.
On remarque qu'un tel type de système ne permet pas de créer de nombreux liens entre les données. Il n'y a pas de passages directs entre « fils » par exemple.
Ce modèle n'est plus utilisé couramment actuellement.

2.1.2.2 Modèle réseau ou navigationnel

C'est le modèle utilisé par les systèmes CODASYL. On dispose alors de liens cycliques entre les enregistrements. Le **CODASYL** (sigle de *Conference on Data Systems Languages*, en français « Conférence sur les langages de systèmes de traitement de données ») est l'organisme américain de codification des systèmes de bases de données. Il a publié en 1959 les spécifications du langage COBOL. Par la suite, ses travaux entre 1974 et 1981 ont donné naissance au modèle navigationnel de SGBD (opposé au modèle hiérarchique largement promu par IBM).



Dans ce modèle le père a l'adresse du premier fils et chaque fils a l'adresse du frère qui le suit et celle de son père. Le dernier frère a uniquement l'adresse de son père.
Les liens entre enregistrements sont déjà beaucoup plus nombreux et complexes. L'accès aux données se trouve facilité. Avec un tel type de modèle, les entités sont reliées entre elles grâce à des pointeurs logiques qui se trouvent au sein des enregistrements.

2.1.2.3 Modèle relationnel

Le modèle relationnel est très performant et le plus utilisé à l'heure actuelle. Ce modèle sera étudié plus en détail notamment dans le point 2. Ce modèle se caractérise par le fait que les entités ainsi que les liens sont représentés sous forme de tables.
Exemple : Une personne serait modélisée sous forme d'une ligne au sein d'une table de la base, chaque élément de la ligne correspondant à une caractéristique de la personne (nom, prénom...).

Note : il existe également le modèle objet.

2.1.3 Niveau physique

Le schéma physique correspond à une représentation des données sur le support de stockage qui sera utilisé. On travaille alors avec les éléments suivants : les fichiers, les tables, les enregistrements,...

2.1.4 Niveau externe

Un SGBD est en général utilisé par plusieurs utilisateurs.
Le schéma conceptuel global des données représente la totalité des informations connues par le système. Cependant, un poste de travail n'utilise souvent qu'une petite partie de toutes les informations stockées dans la base de données. Au niveau externe, on introduit alors le concept de « vue », c'est à dire que l'on montre à l'utilisateur uniquement la portion du schéma conceptuel qui le concerne.
Les différentes vues sont décrites au niveau de schémas externes, chaque schéma externe traduit un type d'utilisation de la base de données.

2.2 Le modèle relationnel

2.2.1 Notion de relation

Une relation peut se représenter sous la forme d'une table (ou tableau).

Exemple :

NOM_F	PRENOM	AGE
...	...	...

Cette table est construite à partir de trois domaines :

- le domaine des noms;
- le domaine des prénoms;
- le domaine des âges.

Chaque ligne résulte de l'association des trois attributs : NOM_F, PRENOM, AGE.

2.2.2 Schéma relationnel

Le schéma d'une relation est constitué :

- du nom de la relation;
- de la liste des attributs de cette relation.

NB : Une relation pourra donc être représentée : soit sous la forme d'un tableau;
soit sous la forme d'un schéma.

Au niveau d'une relation, il faudra également déterminer une clé (un identifiant) qui permettra de distinguer les enregistrements.

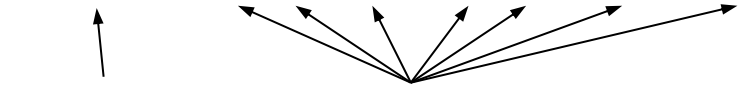
Exemple :

Supposons que l'on veuille créer une base de données reprenant les caractéristiques de différents monuments de différentes villes du monde entier.
Chaque monument sera caractérisé par son nom, de plus on aimerait disposer pour chaque monument :

- du tarif de visite;
- du numéro;
- de la rue;
- de la ville;
- de la population de la ville concernée;
- du pays.

On pourrait alors considérer la relation dont le schéma figure ci-après :

Monument (nom_m, tarif, numéro, rue, ville, population, pays).



nom de la relation **attributs**

Avec une telle relation, si on a trois monuments situés dans une même ville, on aura une redondance des informations : « Pays » et « Population ».

Exemple (présentation de la relation sous la forme d'un tableau) :

NOM_M	TARIF	NUMERO	RUE	VILLE	POPULATION	PAYS
Arc de T.	...	...	...	Paris	11000000	France
Tour Eiffel	...	...	...	Paris	11000000	France
Le Louvre	...	...	...	Paris	11000000	France
Atomium	...	...	...	Bruxelles	1000000	Belgique

Afin d'éviter une telle redondance d'informations, on va scinder la relation précédemment envisagée en deux relations :

Ville (nom_ville, population, pays)

Monument (nom_m, tarif, numéro, rue, nom_ville)

Table « Monument » :

NOM_M	TARIF	NUMERO	RUE	NOM_VILLE
...	...	...	...	...

Table « Ville » :

NOM_VILLE	POPULATION	PAYS
...	...	...

Dans le cas présent, on suppose que NOM_M et NOM_VILLE sont susceptibles de jouer le rôle de clé primaire. Pratiquement, dans certains cas, il faudrait travailler avec d'autres champs. Il est en effet possible d'avoir différentes villes ou différents monuments qui portent le même nom.

2.2.3 Quelques définitions

- **Attribut :**
  L'attribut désigne un champ ou une propriété du lot d'informations.

- **Relation (ou table):**
  C'est l'ensemble des lignes toutes caractérisées par les mêmes attributs.

  Exemple :

PERSONNE		
NOM_F	PRENOM	DN
Dupont	René	10/06/1987
...	...	...
...	...	...

- **Tuple (ou ligne) :**
  C'est une ligne du tableau.
  C'est une occurrence du lot d'informations.

NB : Une relation est donc un ensemble de tuples. De plus, une relation ne peut pas comporter deux tuples identiques.

- **Arité :**
  L'arité d'une relation est le nombre de ses attributs. Exemple : la relation R(A1, A2, ..., An) est d'arité n.
  L'arité d'une relation correspond au nombre de colonnes de la table.

- **Cardinalité d'une relation :**
  La cardinalité d'une relation correspond au nombre de tuples de la relation autrement dit au nombre de lignes de la table.

2.2.4 Algèbre relationnelle

Dans ce point, nous allons examiner les opérateurs permettant d'agir sur les relations.

2.2.4.1 Somme (ou union)

NB : Pour effectuer la somme de deux relations, celles-ci doivent avoir même arité. De plus, pour que l'union ait un sens, les deux relations doivent avoir des attributs de même type.

Exemple :
Soit la relation R1(nom client, code client, adresse client, n° secteur) qui définit la liste des clients visités par le vendeur V1 pendant une période donnée.

NOM CLIENT	CODE CLIENT	ADRESSE CLIENT	N° SECTEUR
DUPONT	C1212	BORDEAUX	17
DURAND	C1217	TALENCE	17
MARTIN	C1218	BEGLES	17
RAOUL	C1120	BORDEAUX	17

Soit la relation R2(nom client, code client, adresse client, n° secteur) qui définit la liste des clients visités par le vendeur V2 sur la même période.

NOM CLIENT	CODE CLIENT	ADRESSE CLIENT	N° SECTEUR
DUPONT	C1212	BORDEAUX	17
DUBOIS	C1208	BORDEAUX	17
PIERRE	C1202	BORDEAUX	17
RAOUL	C1120	BORDEAUX	17

NB : Supposons que les deux vendeurs V1 et V2 peuvent visiter les mêmes clients et que ce sont les seuls vendeurs pour le secteur 17.

Soit la relation R = R1 U R2

NOM CLIENT	CODE CLIENT	ADRESSE CLIENT	N° SECTEUR
DUPONT	C1212	BORDEAUX	17
DURAND	C1217	TALENCE	17
MARTIN	C1218	BEGLES	17
RAOUL	C1120	BORDEAUX	17
DUBOIS	C1208	BORDEAUX	17
PIERRE	C1202	BORDEAUX	17

Remarque :

La relation R est obtenue en faisant la somme des lignes de R1 et de R2 cependant les lignes figurant à la fois dans R1 et dans R2 ne se trouvent qu'une fois dans la relation R. Il faut, en effet, se rappeler que l'on ne peut pas avoir deux lignes identiques au sein d'une table.

La relation R contient tous les clients du secteur 17 qui ont été visités sur une période donnée.

2.2.4.2 Différence

Si on considère les deux tables (R1 et R2) présentées précédemment, R1-R2 correspondrait à la table suivante :

NOM CLIENT	CODE CLIENT	ADRESSE CLIENT	N° SECTEUR
DURAND	C1217	TALENCE	17
MARTIN	C1218	BEGLES	17

Généralisation :

Soit les relations R et S.

R-S est constituée de l'ensemble des lignes de R qui n'appartiennent pas à S.

2.2.4.3 Produit cartésien

Soit deux relations R et S :

R est d'arité k1.
S est d'arité k2.

Le produit cartésien de R et de S (R X S) donnera lieu à une relation d'arité k1 + k2. Les tuples de la relation obtenue sont le résultat de la concaténation des tuples de R avec ceux de S.

Soit la relation R suivante :

A	B	C
a	b	c
d	a	f
c	b	d

Soit la relation S suivante :

D	E	F
b	g	a
d	a	f

R X S :

A	B	C	D	E	F
a	b	c	b	g	a
a	b	c	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

Exemple :

Soit les deux relations suivantes :
R1 : liste des clients du secteur 17.
R2 : liste des vendeurs du secteur 17.

R1 :

NOM CLIENT	CODE CLIENT	ADRESSE CLIENT	N° SECTEUR
DUPONT	C1212	BORDEAUX	17
DURAND	C1217	TALENCE	17
MARTIN	C1218	BEGLES	17
RAOUL	C1120	BORDEAUX	17

R2 :

NOM VENDEUR	CODE VENDEUR
X	JFD
Y	RF

R1 X R2 :

NOM CLIENT	CODE CLIENT	ADRESSE CLIENT	N° SECTEUR	NOM VENDEUR	CODE VENDEUR
DUPONT	C1212	BORDEAUX	17	X	JFD
DUPONT	C1212	BORDEAUX	17	Y	RF
DURAND	C1217	TALENCE	17	X	JFD
DURAND	C1217	TALENCE	17	Y	RF
MARTIN	C1218	BEGLES	17	X	JFD
MARTIN	C1218	BEGLES	17	Y	RF
RAOUL	C1120	BORDEAUX	17	X	JFD
RAOUL	C1120	BORDEAUX	17	Y	RF

La relation obtenue définit la liste des clients du secteur 17 et des vendeurs pouvant les visiter.

2.2.4.4 Intersection

Soit R1 :

DATE	VENDEUR	SECTEUR	CLIENT	CODE CLIENT	ADRESSE	VENTE
2/02	JFD	17	DURAND	C1217	TALENCE	7850
2/02	RF	17	DUPONT	C1212	BORDEAUX	8720
4/02	LM	17	DUPONT	C1212	BORDEAUX	7632
5/02	LM	17	DURAND	C1217	TALENCE	9475

Soit R2 :

DATE	VENDEUR	SECTEUR	CLIENT	CODE CLIENT	ADRESSE	VENTE
2/02	RF	17	DUPONT	C1212	BORDEAUX	8720
3/02	RF	19	JACQUES	C1435	BERGERAC	7625

R1 ∩ R2 :

DATE	VENDEUR	SECTEUR	CLIENT	CODE CLIENT	ADRESSE	VENTE
2/02	RF	17	DUPONT	C1212	BORDEAUX	8720

La relation résultant de l'intersection est constituée des lignes communes aux deux relations qui servent de base à l'intersection.

2.2.4.5 Jointure

La jointure des deux relations R et S suivant la condition i * j (* est un opérateur: =, < >, >, <) est l'ensemble des lignes du produit cartésien R X S qui satisfont à la condition i * j (i est un attribut de R et j est un attribut de S).

Notation : R X S
i * j

Exemple :

Soit la relation R :

A	B	C
1	2	3
4	5	6
7	8	9
0	1	2

Soit la relation S :

D	E
3	1
6	2

R X S
B < D

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2
0	1	2	3	1
0	1	2	6	2

R X S
C = D

A	B	C	D	E
1	2	3	3	1
4	5	6	6	2

2.2.4.6 Équijointure

L'équijointure est un cas particulier de jointure dans lequel l'opérateur est l'égalité.

Exemple :

Soit R :

A	B	C
1	2	3
3	2	1
2	3	4
2	2	5
4	5	6
7	8	9
7	3	2

Soit S :

D	E
2	8
4	7
3	9
3	8

R X S
B = D

A	B	C	E
1	2	3	8
3	2	1	8
2	3	4	9
2	3	4	8
2	2	5	8
7	3	2	9
7	3	2	8

Note : Au niveau de la relation obtenue, on évite d'écrire deux colonnes identiques.

Exemple concret :

Soit la relation VENDEUR :

CODE VENDEUR	NOM VENDEUR	SECTEUR
JFD	DUBOURG	17
RF	FOURNIER	17
RF	FOURNIER	19
LM	MARTY	19

Soit la relation CLIENT :

SECTEUR	CODE CLIENT	NOM CLIENT	ADRESSE
17	C1212	DUPONT	BORDEAUX
17	C1217	DURAND	TALENCE
19	C1218	MARTIN	LIMOGES
19	C1220	JACQUES	BERGERAC
17	C1230	RAOUL	BORDEAUX

Soit la relation TOURNÉE qui serait le résultat de l'opération suivante :

VENDEUR X CLIENT
VENDEUR.SECTEUR = CLIENT.SECTEUR

CODE VENDEUR	NOM VENDEUR	SECTEUR	CODE CLIENT	NOM CLIENT	ADRESSE
JFD	DUBOURG	17	C1212	DUPONT	BORDEAUX
JFD	DUBOURG	17	C1217	DURAND	TALENCE
JFD	DUBOURG	17	C1230	RAOUL	BORDEAUX
RF	FOURNIER	17	C1212	DUPONT	BORDEAUX
RF	FOURNIER	17	C1217	DURAND	TALENCE
RF	FOURNIER	17	C1230	RAOUL	BORDEAUX
RF	FOURNIER	19	C1218	MARTIN	LIMOGES
RF	FOURNIER	19	C1220	JACQUES	BERGERAC
LM	MARTY	19	C1218	MARTIN	LIMOGES
LM	MARTY	19	C1220	JACQUES	BERGERAC

2.2.4.7 Jointure naturelle

Dans le cas où R et S ont des attributs communs, l'égalité est définie implicitement sur ces attributs et on note simplement R X S.
NB : Une nouvelle fois, les attributs communs à R et S ne sont pas dupliqués dans la relation résultat.

Exemple :

Soit la relation R suivante :

A	B	C
a	b	c
a	b	e
a	c	d
a	d	e
b	d	e

Soit la relation S :

B	C	D
b	c	d
b	c	f
c	e	d
d	e	b

R X S :

A	B	C	D
a	b	c	d
a	b	c	f
a	d	e	b
b	d	e	b

NB : Il s'agit en fait de : R X S
R.B = S.B
R.C = S.C

Le deuxième exemple présenté au niveau de l'équijointure est également un cas de jointure naturelle car elle est effectuée sur des attributs de même désignation (secteur) dans les relations VENDEUR et CLIENT.

3 Compléments théoriques concernant les notions de tables, de colonnes, de contraintes, ...

3.1 Table, ligne et colonne

Une table contient une suite de lignes stockées sur un support externe, généralement un disque. Une ligne est elle-même une suite d'une ou plusieurs valeurs, chacune étant d'un type déterminé. D'une manière générale, une ligne regroupe des informations concernant un individu, un objet, un événement, etc., c'est-à-dire un concept du monde réel que nous appellerons parfois une entité ou un fait.

Exemple de ligne :

GREGOIRE	PAUL	ARLON

La ligne figurant ci-dessus contient des informations sur une personne. On y trouve trois valeurs représentant respectivement son NOM, son PRENOM et la LOCALITE où elle réside.
Cette ligne représente le fait suivant : il existe une personne dont le nom est GREGOIRE et le prénom est PAUL résidant à ARLON.

Toutes les lignes présentes dans une table ont même format ou structure. Cette propriété signifie que dans la table PERSONNE dont la ligne précédente est extraite, toutes les lignes sont constituées d'une valeur de NOM, d'une valeur de PRENOM, d'une valeur LOCALITE. Ci-dessous figure un exemple de contenu de la table PERSONNE. Cette table contient 4 lignes de format identique.

PERSONNE		
NOM_F	**PRENOM**	**LOCALITE**
Domingez	Sandra	Autelbas
Dupont	René	Arlon
Ney	Corine	Aubange
Durand	Paul	Virton

L'ensemble des valeurs de même type correspondant à une même propriété des entités décrites s'appelle une colonne de la table. On parlera d'une colonne NOM_F de la table PERSONNE. « Ney » est la valeur de la colonne NOM_F de la troisième ligne. On définira une colonne par son nom, le type de ses valeurs et une longueur maximale.
Il est possible d'ajouter des lignes à une table et d'en supprimer. Il est possible également de modifier la valeur d'une colonne, d'une ligne, ou plus généralement d'un sous-ensemble des lignes.

3.2 Rôles d'une colonne

On admet qu'une ligne regroupe des informations sur une entité ou un fait du monde réel. Dans cette optique, la valeur d'une colonne représente une propriété de cette entité.

FOURNISSEUR		
NUMF	**NOMF**	**VILLEF**
152	MERCIER	ANVERS
174	CHARLES	LIEGE
81	DUMONT	LIEGE
376	RENIER	LILLE
259	CHARLES	LIEGE
46	GARCIN	MONS

PIECE	
NUMP	TYPE
14	BOULON_A
57	ECROU
15	BOULON_B

LIVRER		
NUMF	NUMP	PRIX
152	15	46
174	57	32
152	14	62
81	14	65
81	15	48
152	57	34
46	15	46
46	57	32

Les trois tables ci-dessus représentent une base de données.

- La première table, de nom FOURNISSEUR décrit des fournisseurs. Elle spécifie le numéro (NUMF), le nom (NOMF) du fournisseur ainsi que sa ville de résidence (VILLEF).
- La deuxième table, nommée PIECE, décrit des pièces de rechange, caractérisées par leur numéro (NUMP) et leur type (TYPE).
- La troisième table (LIVRER) représente les possibilités de livraison. Chaque possibilité de livraison est caractérisée par le numéro du fournisseur (NUMF) ainsi que par le numéro de la pièce livrable (NUMP) et le prix (PRIX) auquel ce fournisseur livre la pièce.

Sur base de l'exemple de base de données exposé précédemment, on peut distinguer trois types de colonnes.

Premier type
Ce premier type de colonne permet d'identifier une entité (autrement dit une ligne) au sein de la table. Au sein de la table FOURNISSEUR c'est la colonne NUMF qui joue ce rôle et au niveau de la table PIECE c'est la colonne NUMP.
Une telle colonne est appelée l'**identifiant** de la table. Si on déclare que NUMF est l'identifiant de la table FOURNISSEUR, c'est imposer qu'à tout instant les lignes de cette table doivent avoir des valeurs distinctes de NUMF.

Deuxième type
Une colonne du deuxième type est caractérisée par le fait qu'elle contient différentes valeurs faisant référence à l'identifiant d'une autre table. Une telle colonne est appelée **colonne de référence**, ou **clé étrangère**.
La table LIVRER contient deux clés étrangères :

- NUMF qui fait référence à la table FOURNISSEUR.
- NUMP qui fait référence à la table PIECE.

C'est grâce à ces clés étrangères que l'on peut créer des relations entre les tables. Dans notre exemple, pour chaque livraison possible représentée par une ligne de la table LIVRER, il est possible de connaître les informations concernant le fournisseur (via NUMF) et celles qui concernent la pièce (via NUMP).

Troisième type
Ce troisième type de colonne ne joue d'autre rôle que celui d'apporter une information complémentaire sur l'entité (exemple : PRIX).

À partir de ces éléments de base, nous mettrons en évidence quatre concepts importants : les identifiants multicomposants, les identifiants primaires, les contraintes référentielles et les colonnes facultatives.

a) Les identifiants multicomposants
Rien n'interdit que l'identifiant d'une table soit constitué de plus d'une colonne. On pourrait ainsi imposer que le couple (NUMF, NUMP) soit l'identifiant de la table LIVRER.

b) Les identifiants primaires
Rien n'interdit non plus qu'on impose plus d'un identifiant à une table. Par exemple, la table qui reprend les informations signalétiques d'une population d'assurés sociaux pourrait inclure, entre autres, une colonne NUMERO_INSCRIPTION et une colonne NUMERO_CARTEID, chacune constituant un identifiant. Parmi les identifiants d'une table, l'un est choisi comme le plus représentatif. Il sera déclaré **identifiant primaire**, les autres étant les identifiants secondaires de la table. Toute table possède un identifiant primaire (c'est vivement conseillé) et un nombre quelconque (éventuellement nul) d'identifiants secondaires.

c) Les contraintes référentielles
Une valeur d'une clé étrangère constituée de la colonne CB d'une table B est destinée à désigner une ligne d'une table A. Concrètement, on peut imposer que toute valeur de CB dans B soit une valeur effectivement prise par l'identifiant primaire de la table A. On imposerait par exemple que toute valeur de NUMP dans LIVRER soit présente dans PIECE comme valeur de la colonne NUMP, et que toute valeur de NUMF dans LIVRER se retrouve comme valeur NUMF de la table FOURNISSEUR. Cette propriété porte le nom de contrainte référentielle. Si NUMF de LIVRER fait l'objet d'une contrainte référentielle, il est interdit d'introduire dans cette table une ligne telle que (NUMF : 93, NUMP: 57, PRIX: 32), puisqu'il n'existe dans FOURNISSEUR aucune ligne dont la clé primaire NUMF ait la valeur 93.

d) Les colonnes facultatives
Il se peut qu'une information ne soit pas connue au moment où on introduit la description d'une entité dans une table. Par exemple, on apprend que le fournisseur 259 peut désormais livrer des pièces « 15 », mais on n'en connaît pas encore le prix. On admet alors d'introduire dans la table LIVRER la ligne incomplète suivante : (NUMF : 259, NUMP: 15, PRIX: -). On dira que la valeur de PRIX de cette ligne est inconnue. Cette latitude donnée à une colonne d'accepter l'absence de valeur pour certaines lignes n'est pas valable pour toute colonne. Par exemple, on peut décider que le nom d'un fournisseur est indispensable, et donc que toute ligne de la table FOURNISSEUR doit avoir une valeur NOMF valable. La colonne PRIX sera déclarée **facultative**, tandis que la colonne NOMF sera **obligatoire**.
D'une façon générale, on peut dire que les composants de tout identifiant primaire sont obligatoires.

3.3 Structure et contenu d'une base de données

On distingue deux classes de composants dans une base de données : son schéma et son contenu. Le schéma d'une base de données en définit la structure en termes de tables, de colonnes, d'identifiants primaires et secondaires, et de contraintes référentielles.

Soit la table FOURNISSEUR. Son schéma peut se représenter de deux façons :

Soit FOURNISSEUR(NUMF, NOMF, VILLEF)

Soit :

FOURNISSEUR		
NUMF	NOMF	VILLEF

Le contenu d'une base de données est en général très volumineux (plusieurs millions de lignes) et est susceptible d'évoluer constamment. On ne pourra donc pas toujours le représenter de façon exhaustive. Notre intérêt se portera essentiellement sur le schéma des tables. Les schémas sont généralement relativement stables dans le temps. En effet, on ne modifie la structure d'une base de données que lorsque la structure de la réalité à représenter évolue.

3.4 Conventions de représentation

Une table et ses colonnes sont représentées par un cartouche contenant le nom de la table et celui de chaque colonne. Les colonnes sont placées dans un ordre quelconque. Cependant, afin de faciliter la compréhension, on veillera autant que possible à disposer de façon contiguë les colonnes d'un même identifiant de même que celles de chaque clé étrangère, et à placer à l'extrême gauche les colonnes de l'identifiant primaire.

Exemple :

LIVRER		
NUMF	NUMP	PRIX

On soulignera d'un trait continu les noms des colonnes de l'identifiant primaire, et d'un trait pointillé ceux de chaque identifiant secondaire.

LIVRER		
NUMF	NUMP	PRIX

Le nom d'une colonne facultative sera mis entre parenthèses.

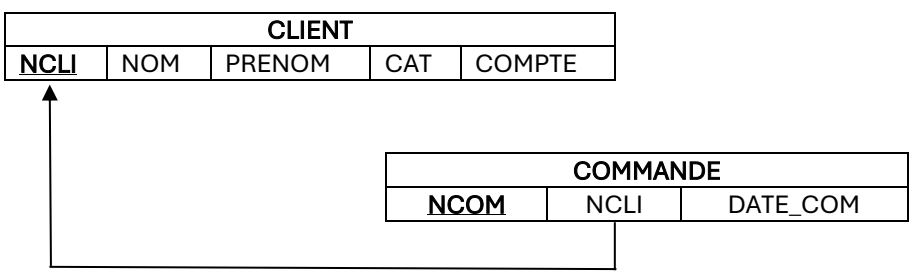
LIVRER		
NUMF	NUMP	(PRIX)

Une clé étrangère de plus d'une colonne de référence sera représentée par une accolade.

ARTICLE			
NUMA	NUMF	ANNEE	SAISON



Une contrainte référentielle sera représentée par une flèche qui part du nom de la colonne de référence (ou de l'accolade en cas de groupe de référence), et qui pointe vers le cartouche de la table référencée.



3.5 Un exemple de base de données relationnelle

Partons de l'exemple suivant :

dbo_tbl_client_cli

cli_ncli	cli_titre	cli_nom	cli_prenom	cli_rue	cli_cp	cli_localite	cli_cat	cli_compte
B062	Madame	Goffin	Jacqueline	Rue de la Gare 72	5000	Namur	B2	-3200
B112	Monsieur	Hansenne	Albert	Rue A. Dumont 23	5300	Andenne	C1	1250
B332	Monsieur	Monti	Paul	Rue Neuve 13	4970	Stavelot	B2	0
B512	Monsieur	Gillet	Grégoire	Rue de la Gare 14	4000	Liège	B1	-8700
C003	Monsieur	Avron	François	Chemin de la Cure 8	4000	Liège	B1	-1700
C123	Monsieur	Mercier	Jean	Rue Lemaître 25	5000	Namur	C1	-2300
C400	Madame	Ferard	Anne	Rue du Tertre 65	5300	Andenne	B2	350
D063	Madame	Nadoy	Jacqueline	Boulevard de la Poste 1	4000	Liège	C3	-2250
F010	Madame	Toussaint	Corine	Rue Godefroid 5	5300	Andenne	C1	0
F011	Monsieur	Poncelet	Fernando	Clos des Erables 17	4000	Liège	B2	0
F400	Monsieur	Jacob	Daniel	Chemin du Moulin 78	3000	Louvain	C2	0
K111	Madame	Vanbist	Sylvie	Rue Florimont 81	4540	Amay	B1	720
K729	Monsieur	Neuman	Herbert	Rue Bransart 40	4000	Liège	C3	0
L422	Madame	Franck	Albertine	Rue de Wépion 60	5000	Namur	C1	0
S127	Monsieur	Vanderka	René	Avenue des Roses 3	5000	Namur	C1	-4580
S712	Monsieur	Guillaume	Albert	Rue du Lac 14a	5100	Wépion	B1	0

dbo_tbl_commande_com

com_ncom	com_ncli	com_date
178	K111	2025-06-12
179	C400	2025-06-12
182	S127	2025-06-14
184	C400	2025-06-27
185	F011	2025-07-02
186	C400	2025-07-02
188	B512	2025-07-03

dbo_tbl_lignecom_lig

lig_ncom	lig_npro	lig_prix	lig_qcom
178	CS464	6	25
179	CS262	5,5	60
179	PA60	5	20
182	PA60	5	30
184	CS464	6	120
184	PA45	4	20
185	CS464	6	260
185	PA60	5	15
185	PS222	15	600
186	PA45	4	3
188	CS464	6	180
188	PA45	4	22
188	PA60	5	70
188	PH222	12	92

dbo_tbl_produit_pro

pro_npro	pro_libelle	pro_prix	pro_qstock
CS262	Chevron en sapin 300 x 4 x 6 cm	5,5	45
CS264	Chevron en sapin 300 x 6 x 6 cm	6,8	2690
CS464	Chevron en sapin 400 x 4 x 6 cm	6	450
PA45	Pointes en acier 45 (1 kg)	4	580
PA60	Pointes en acier 60 (1 kg)	5	134
PH222	Planche en hêtre 200 x 2 x 6 cm	12	782
PS222	Planche en hêtre 200 x 3 x 6 cm	15	1220

Table tbl_client_cli
Chaque ligne décrit un client, les colonnes décrivent successivement le numéro du client (cli_ncli), son nom (cli_nom), son prénom (cli_prenom)... L'identifiant primaire est constitué de cli_ncli.

Table tbl_produit_pro
Chaque ligne décrit un produit, les colonnes décrivent successivement le numéro du produit (pro_npro), son libellé (pro_libelle), son prix unitaire (pro_prix) et la quantité restant en stock (pro_qstock). pro_npro est l'identifiant primaire.

Table tbl_commande_com
Chaque ligne décrit une commande passée par un client, les colonnes décrivent successivement le numéro de la commande (com_ncom), le numéro du client qui a passé la commande (com_ncli), et la date de la commande (com_date). com_ncom est l'identifiant primaire de la table.
com_ncli est une clé étrangère vers la table tbl_client_cli.

Table tbl_lignecom_lig
Chaque ligne représente une ligne d'une commande, les colonnes décrivent successivement le numéro de la commande à laquelle la ligne appartient (lig_ncom), le numéro du produit commandé (lig_npro), la quantité commandée (lig_qcom) et le prix appliqué (lig_prix). L'identifiant primaire est constitué de lig_ncom et lig_npro. lig_ncom et lig_npro sont en outre deux clés étrangères respectivement vers les tables tbl_commande_com et tbl_produit_pro.

3.6 Modification des données et contraintes d'intégrité

Les propriétés structurelles (identifiant, contrainte référentielle, colonne obligatoire/facultative) associées aux données doivent être respectées à tout instant, et constituent donc des contraintes imposées aux opérations de modification de ces données. Ajouter une ligne, supprimer une ligne ou modifier une valeur de colonne d'une ligne ne peut être effectué que si ces propriétés sont toujours respectées par les données résultant de l'opération. Si ces propriétés sont violées, on dira que les données ont perdu leur intégrité. Ces propriétés constituent dès lors des contraintes d'intégrité. Nous examinerons brièvement l'impact de ces contraintes sur les opérations de modification du contenu d'une base de données.

3.6.1 Les identifiants

Un identifiant constitue une contrainte d'unicité imposant qu'à tout instant les lignes d'une table possèdent des valeurs distinctes pour une ou plusieurs colonnes. Il ne peut exister, à aucun moment, plus d'une ligne de tbl_client_cli ayant la même valeur de NCLI.

- Création d'une ligne : il ne peut pas exister une autre ligne possédant la même valeur au niveau de l'identifiant.
- Suppression d'une ligne : pas de contrainte.
- Modification de l'identifiant d'une ligne : il ne peut pas exister une autre ligne possédant la nouvelle valeur de l'identifiant.

3.6.2 Les contraintes référentielles

Une contrainte référentielle précise que certaines colonnes d'une table, appelées clé étrangère, doivent à tout instant, pour chaque ligne, contenir des valeurs que l'on retrouve comme identifiant primaire d'une ligne dans une autre table. C'est ainsi que la table tbl_lignecom_lig est soumise à deux contraintes référentielles. La première indique que toute valeur de lig_ncom doit identifier une ligne de tbl_commande_com. La seconde indique que toute valeur de lig_npro doit identifier une ligne de tbl_produit_pro.

Les règles qui régissent les opérations de modification des données sont plus complexes que pour les autres contraintes. Nous raisonnerons sur les tables tbl_client_cli et tbl_commande_com, cette

dernière faisant l'objet d'une contrainte référentielle, puisque la valeur de com_ncli de toute ligne de tbl_commande_com doit être présente dans la colonne cli_ncli d'une ligne de tbl_client_cli (concrètement, toute commande doit appartenir à un client enregistré). On ignorera dans cette analyse l'impact des autres contraintes référentielles du schéma sur les opérations.

Création d'une ligne dans la table tbl_commande_com
La valeur de com_ncli de cette ligne doit être présente dans la colonne cli_ncli d'une ligne de la table tbl_client_cli (si com_ncli de tbl_commande_com avait été déclarée facultative alors la valeur de com_ncli de la ligne pourrait être absente).

Suppression d'une ligne dans la table tbl_commande_com
Effectuée sans restriction (si on veut être complet il faut dire que la suppression d'une ligne de la table tbl_commande_com peut avoir un impact sur tbl_lignecom_lig, pour simplifier le raisonnement nous négligeons cet aspect dans le cas présent).

Modification de la valeur de com_ncli dans la table tbl_commande_com
La nouvelle valeur de com_ncli de la ligne doit être présente dans la colonne cli_ncli d'une ligne de tbl_client_cli (si com_ncli de tbl_commande_com avait été déclarée facultative alors on pourrait supprimer la valeur de com_ncli de la ligne).

Création d'une ligne CLIENT
Effectuée sans restriction.

Suppression d'une ligne CLIENT
Le problème se pose lorsque ce client possède des commandes, il existe alors des lignes dans tbl_commande_com qui référencent la ligne de tbl_client_cli à supprimer. Il existe deux comportements possibles qui laissent la base de données dans un état cohérent.

- Le premier consiste à refuser la suppression de la ligne de tbl_client_cli afin d'éviter de laisser dans la base de données des lignes de tbl_commande_com orphelines.
- La deuxième réaction consiste à supprimer non seulement la ligne de tbl_client_cli, mais aussi toutes les lignes de tbl_commande_com qui la référencent.

Modification de cli_ncli d'une ligne dans la table tbl_client_cli
Si aucune ligne de tbl_commande_com ne référence cette ligne de tbl_client_cli, la contrainte référentielle n'impose pas de restriction. Si au contraire de telles lignes existent, soit la valeur de cli_ncli est modifiée à la fois dans la ligne de tbl_client_cli et dans les lignes de tbl_commande_com qui lui font référence soit la modification est refusée.

On voit donc que l'impact d'une contrainte référentielle n'est pas unique, et que, selon la réalité à représenter et la politique de gestion des données, on sera amené à choisir l'un ou l'autre des comportements décrits.

3.6.3 Les colonnes obligatoires

Pour toute colonne déclarée obligatoire chaque ligne doit posséder une valeur. Lors des opérations de création et de modification de lignes, cette colonne devra recevoir une valeur.

II Le langage SQL

1 Présentation de la base de données exemple

1.1 Présentation

Un exemple de base de données que nous allons utiliser tout au long de ce chapitre concerne une université fictive nommée « Université Flugle ». Partons de l'hypothèse que le directeur a fait appel à vos compétences pour développer un système d'information automatisé basé sur une architecture client-serveur. Les conventions de nommage utilisées ci-après sont propres à cette université et sont différentes de nos exercices pratiques.

1.2 Analyse des besoins

Les importants besoins du système d'information **(SI)** de l'université comprennent :

- Le stockage, la maintenance et la recherche d'informations qui concernent les étudiants.
- Le stockage, la maintenance et la recherche d'informations qui concernent les enseignants.
- Le stockage, la maintenance et la recherche d'informations qui concernent chaque département.
- Le stockage, la maintenance et la recherche d'informations qui concernent les cours proposés.
- Le stockage, la maintenance et la recherche d'informations qui concernent le calendrier des cours.
- La préparation et l'impression des catalogues de cours semestriels.

À partir de ces besoins, les entités suivantes doivent être créées :

- la table Student qui contient des informations sur chaque étudiant de l'université;
- la table Department qui contient des informations sur chaque département de l'université;
- la table Instructor qui contient des informations sur chaque enseignant de l'université;
- la table Course qui contient des informations sur chaque cours proposé par l'université;
- la table Class Location qui identifie chaque salle disponible pour un cours;
- la table Schedule Type qui identifie chaque type de calendrier, pour prévoir deux réunions par semaine, par exemple;
- la table Schedule Type Détails qui contient les informations sur un type de calendrier particulier, tels que les jours et les heures de réunions;
- la table Class qui contient des informations sur chaque classe programmée tels que le cours, l'enseignant et le calendrier, une classe étant la proposition d'un cours durant un semestre particulier;
- la table Student Schedule identifie les classes pour lesquelles un étudiant s'est engagé.

La prochaine section décrit chaque table en détail.

1.3 Présentation des tables

Table Student

La table Student conserve des informations sur chaque étudiant de l'université. Chacun est identifié par un attribut Student ID et la table est définie de telle manière qu'il ne puisse y avoir deux étudiants possédant le même identifiant. Outre le nom, l'adresse et d'autres informations pertinentes, elle contient un attribut nommé Year, indiquant le nombre d'années passées dans l'établissement, par exemple, première année ou deuxième année. Au vu de cela, on peut constater que cette base de données est quelque peu irréaliste (mais il faut bien se limiter pour avoir un exemple concis). Il est probablement plus important de garder trace des unités accumulées par l'étudiant que du nombre d'années qu'il a passées à l'université. Une autre remarque est que la table n'indique pas le statut de l'étudiant, par exemple, présent, absent ou autorisé à s'absenter. Il n'existe aucun attribut indiquant qu'un étudiant a reçu un degré quelconque. Néanmoins, les attributs suivants sont suffisants pour notre objectif.

Student ID
Last Name
First Name

Middle Initial
Street Address
City
State
Zipcode
Telephone
Fax
Email
Year in School
Clé primaire : Student ID

Table Department
En créant la table Department, vous garantissez que les autres tables se réfèrent à un département de façon cohérente. La clé primaire pour cette table est Department ID. Voici la structure de cette table :

Department ID
Description
Clé primaire : Department ID

Table Instructor
Les informations concernant chaque enseignant sont conservées dans la table Instructor selon la structure ci-après. La clé primaire pour la table est Instructor ID. La table comprend aussi une clé étrangère, Department ID qui pointe sur la table Department.

Instructor ID
Department ID (clé étrangère sur la table Department)
Last Name
First Name
Middle Initial
Position
Telephone
Fax
Email
Clé primaire : Instructor ID

Dans la base de données Flugle, chaque enseignant est assigné à un seul département identifié par l'attribut Department ID. Dans une réelle université, une telle limitation n'existerait pas.

Table Course
Chaque département propose une série de cours. Comme il est possible pour deux départements de proposer un cours portant le même numéro, la clé primaire pour cette table inclut à la fois les attributs Course ID et Department ID. Outre les attributs de la clé primaire, la liste suivante présente tous les attributs caractérisant un cours : un titre, une description, un nombre d'unités et les frais additionnels éventuels.

La clé primaire pour la table Course pourrait seulement être composée de l'attribut Course ID. Mais cela nécessiterait que, pour un même cours, l'attribut Course ID soit différent dans chaque département. Cette condition serait satisfaite si l'on concaténait cet attribut avec Department ID, par exemple, BIO101 pour Biologie 101. Toutefois, l'inconvénient de cette approche est qu'elle ne respecte pas les formes normales, à savoir, Course ID contient maintenant deux éléments d'information, le cours et le département. Bien que cela représente un réel inconvénient, vous trouverez dans la pratique de nombreux exemples de clés primaires constituées de cette manière.

Department ID (clé étrangère sur la table Department)
Course ID
Title
Description

Units
Additional Fees
Clé primaire : Course ID + Department ID

Table Class Location
Cette table permet à l'administrateur de l'université de déterminer l'emplacement approprié où se déroulera une classe en fonction des sièges disponibles. La clé primaire de cette table se compose des attributs Class Building et Class Room comme indiqué ci-après. Dans une réelle application, vous souhaiterez en savoir davantage à propos des locaux, comme les équipements disponibles, tels que les écrans de projection ou les microphones, etc.

Class Building
Class Room
Seating Capacity
Clé primaire : Class Building, Class Room

Table Schedule Type
Cette table est prévue pour décrire différents types de calendriers de classes. Un exemple de type serait un calendrier prévoyant trois réunions par semaine. La clé primaire de cette table est Schedule ID. Voici la structure de la table :

Schedule ID
Description
Clé primaire : Schedule ID

Table Schedule Type Details
La table Schedule Type Details conserve les détails d'un type de calendrier donné, tels que le jour, l'heure de début et la durée de chaque cours. Par exemple, si l'attribut Schedule ID contient un calendrier T10 avec trois cours par semaine à 10 heures du matin, il y aura trois lignes dans cette table. Une ligne caractérisera le cours de lundi à 10h, une autre celui de mercredi à la même heure, et une dernière, celui de vendredi toujours à la même heure. Comme la durée est aussi stipulée pour chaque ligne, la table est en mesure d'enregistrer un type de calendrier dans lequel les cours auraient une durée différente en fonction du jour. Voici la structure de cette table :

Schedule ID (clé étrangère sur la table Schedule Type)
Day
Starting Time
Duration
Clé primaire : Schedule ID, Day

Table Class
La table Class contient des renseignements spécifiques à chaque cours proposé par l'université durant un semestre donné. Vous pouvez penser à une classe comme étant une *instance* d'un cours. Par conséquent, la table doit inclure les attributs Course ID et Department ID. La liste suivante décrit la structure de la table. Chaque classe doit avoir un emplacement et un calendrier et, bien sûr, un enseignant.

Class ID
Department ID (clé étrangère sur la table Course)
Course ID (clé étrangère sur la table Course)
Instructor ID (clé étrangère sur la table Instructor)
Schedule ID (clé étrangère sur la table Schedule Type)
Class Building (clé étrangère sur la table Class Location)
Class Room (clé étrangère sur la table Class Location)
Semester
School Year
Clé primaire : Class ID

Table Student Schedule
La table Student Schedule vise deux objectifs : tout d'abord, elle enregistre les modules qu'un étudiant a obtenus pour chaque classe, ensuite, pour les classes qui ne sont pas encore terminées, elle permet d'identifier les cours faisant partie du calendrier actuel de l'étudiant. Dans une base de données plus réaliste, vous sépareriez probablement les cours qu'un étudiant a suivis et qui sont terminés, de ceux qui font partie du calendrier courant. La clé primaire de la table se compose des attributs Student ID et Class ID

Student ID (clé étrangère sur la table Student)
Class ID (clé étrangère sur la table Class)
Grade
Date Grade Assigned
Clé primaire : Student ID, Class ID

2 Généralités concernant le langage SQL

2.1 Introduction

Le SQL (Structured Query Language) est un langage de requêtes utilisé pour la manipulation des bases de données relationnelles.

Il a été créé au milieu des années 70 par IBM et commercialisé par ORACLE en 1979.

Les instructions sont écrites dans un langage courant (l'anglais !).

Le SQL manipule aussi bien des ensembles d'enregistrements qu'un seul enregistrement, et permet l'utilisation du résultat dans une autre commande.

Le SQL peut être utilisé à tous les niveaux dans la gestion d'une base de données :
- administration système,
- administration de la base,
- développement et application,
- gestion de données.

Tous les utilisateurs de la base ont donc un langage commun.

Ce langage permet d'autre part d'effectuer toutes les opérations :
- interrogation des données,
- ajout, suppression, modification de données,
- gestion des objets (structures),
- gestion de la sécurité d'accès aux données.

2.2 Les objets

- Une base de données relationnelle est composée d'entités logiques et physiques manipulables par le langage SQL.
- Ces entités sont appelées objets SQL.
- On peut regrouper ces objets en différentes catégories en fonction de leur utilité.

2.2.1 Types d'objets

2.2.1.1 La gestion des données

On peut notamment citer:
table Ensemble de lignes (row) et de colonnes (column).
index Colonne ou ensemble de colonnes permettant l'accélération des recherches.
view Requête pouvant être manipulée comme une table (table virtuelle).
rule Règle permettant de vérifier la validité des données.
default Objet permettant de définir une valeur par défaut.

2.2.1.2 Le stockage physique

fichiers de données et les journaux

page La page représente l'unité de travail pour le stockage des données (taille sous SQL Server: 8 Ko).

extension Une extension contient 8 pages de données (64 Ko). Dans une page seront stockées les lignes de la table ou de l'index, à l'exception des champs de type *text* et *image*.

2.2.1.3 Le stockage d'instructions

schema Regroupement logique des objets de la base.

procedure Ensemble de code procédural nommé.

function Ensemble de code procédural nommé retournant une valeur.

trigger Ensemble de code procédural associé à une table et à un événement.

2.2.1.4 La gestion des utilisateurs

role Ensemble d'utilisateurs ayant certains privilèges sur certains objets de la base. Différents types de rôles existent comme cela sera étudié ultérieurement.

user Utilisateur pouvant se connecter et accéder aux ressources de la base.

2.2.2 Dénomination

Voici quelques informations concernant la dénomination des objets créés par les utilisateurs du SGBDR SQL Server.

Les noms peuvent comporter de 1 à 128 caractères.
Les noms des identificateurs peuvent commencer par un caractère alphanumérique ou par _ ; avec SQL Server, ils peuvent contenir pratiquement tous les caractères. **Cependant, il est vivement conseillé de se limiter aux caractères suivants : lettres non accentuées, chiffres, caractère de soulignement _**

2.3 Les catégories d'instructions

Les instructions SQL sont regroupées en catégories en fonction de leur utilité et des entités manipulées.

2.3.1 Le DDL / LDD (Data Definition Language / Langage de définition des données)

Il permet de gérer les « structures » des objets :

- Créer, modifier, supprimer des objets.
- Autoriser ou interdire l'accès aux données.

On peut notamment citer les instructions suivantes : CREATE, ALTER, DROP, GRANT, REVOKE, RENAME, TRUNCATE.

2.3.2 Le DML / LMD (Data Manipulation Language / Langage de manipulation des données)

Il permet la gestion des données des objets existants :

- Ajout, suppression, modification des lignes.
- Visualisation du contenu des tables.
- Consultation du dictionnaire des données.
- Verrouillage des tables.

Quelques instructions du DML: INSERT, UPDATE, DELETE, SELECT.

2.3.3 Le TCL (Transaction Control Language / Langage de contrôle des transactions)

Il gère les modifications faites par le DML :

- Caractéristiques des transactions.
- Validation, annulation des modifications.

Quelques instructions du Transaction Control : COMMIT TRANSACTION, SAVE TRANSACTION, ROLLBACK TRANSACTION.

3 Description et manipulation des objets

3.1 L'instruction CREATE TABLE

3.1.1 Présentation

Cette opération produit une table vide (c'est à dire sans lignes). On spécifie le nom de la table et la description de ses colonnes. On précisera pour chaque colonne son nom, le type de ses valeurs.

Exemple :

```
CREATE TABLE tbl_client_cli (
        cli_ncli char(4),
        cli_nom_f varchar(35),
        cli_rue varchar(45),
        cli_localite varchar(25)
);
```

tbl_client_cli			
cli_ncli	**cli_nom_f**	**cli_rue**	**cli_localite**
……	……	……	……

3.1.2 Types de données

Catégories de types de données SQL Server :

Catégorie	But
Numérique	Stocke des valeurs numériques avec ou sans décimales.
Monétaire	Stocke des valeurs numériques avec des décimales. Employé spécifiquement pour des valeurs monétaires avec jusqu'à quatre décimales.
Date et heure	Stocke des informations de date et/ou d'heure et permet des contraintes chronologiques particulières, comme le refus d'un 30 février…
Caractère	Stocke des valeurs fondées sur des caractères, de longueur fixe ou variable.
Binaire	Stocke des données dans une représentation binaire stricte (0 ou 1).
But spécial	Types de données complexes qui nécessitent une manipulation particulière, comme des documents XML ou des identifiants globalement uniques, …

3.1.2.1 Types de données numériques exacts

Type de données	Stockage	Plage de valeurs	But
bigint	8 octets	-2^{63} à $2^{63}-1$	Stocke de très grands nombres entiers positifs ou négatifs
int	4 octets	-2^{31} à $2^{31}-1$	Stocke des nombres entiers positifs ou négatifs
smallint	2 octets	-32768 à 32767	Stocke des nombres entiers positifs ou négatifs
tinyint	1 octet	0 à 255	Stocke une petite plage de nombres entiers positifs

Type de données	Stockage	Plage de valeurs	But
decimal(p,s) *p est compris entre 1 et 38, 18 par défaut* *s est compris entre 0 et p, 0 par défaut*	5 à 17 octets selon la précision	-10^{38} à $10^{38} - 1$	Stocke des nombres décimaux avec jusqu'à 38 chiffres
numeric(p,s)	5 à 17 octets selon la précision	-10^{38} à $10^{38} - 1$	Equivalent à *decimal* et interchangeable avec lui

3.1.2.2 Types de données numériques à virgule flottante

Type de données	Stockage	Plage de valeurs	But
float(p) *p est compris entre 1 et 53*	4 ou 8 octets	- 1,79E+308 à -2,23E-308 et 2,23E-308 à 1,79E+308	Stocke de grands nombres à virgule flottante qui excèdent la capacité du type *decimal*
real	4 octets	- 3,40E + 38 à -1,18E – 38 et 1,18E - 38 à 3,40E + 38	Encore valide, mais remplacé par *float* pour respecter le standard SQL-92

3.1.2.3 Types de données monétaires

Type de données	Stockage	Plage de valeurs	But
money	8 octets	-922 337 203 685 477,5808 à 922 337 203 685 477,5807	Stocke de grandes valeurs monétaires.
smallmoney	4 octets	-214 748,3648 à 214 748,3647	Stocke de plus petites valeurs monétaires.

3.1.2.4 Types de données « date et heure »

Type de données	Stockage	Plage de valeurs	But
datetime	8 octets	1er janvier 1753 au 31 décembre 9999, avec une précision de 3,33 millisecondes	Stocke de grandes valeurs date/heure
smalldatetime	4 octets	1er janvier 1900 au 6 juin 2079, avec une précision d'une minute	Stocke une plage plus petite de valeurs date/heure

D'autres types de données de type « date/heure » existent : date, time, datetimeoffset, datetime2.

3.1.2.5 Types de données « caractère »

Type de données	Stockage	Plage de valeurs	But
char(n)	1 à 8000 octets	Maximum de 8000 caractères	Type de données non Unicode (ou ANSI) à largeur fixe
nchar(n)	2 à 8000 octets	Maximum de 4000 caractères	Type de données Unicode à largeur fixe
varchar(n)	2 à 8000 octets	Maximum de 8000 caractères	Type de données non Unicode à largeur variable
varchar(max)	Jusqu'à 2 Go	Maximum de 1 073 741 824 caractères	Type de données non Unicode à largeur variable

Type de données	Stockage	Plage de valeurs	But
nvarchar(n)	2 à 8000 octets	Maximum de 4000 caractères	Type de données Unicode à largeur variable
nvarchar(max)	Jusqu'à 2 Go	Maximum de 536 870 912 caractères	Type de données Unicode à largeur variable
text	Jusqu'à 2 Go	Maximum de 2^{31} - 1 caractères	Type de données non Unicode à largeur variable
ntext	Jusqu'à 2 Go	Maximum de 2^{30} - 1 caractères	Type de données Unicode à largeur variable

* ANSI : American National Standard Institute

3.1.2.6 Types de données « binaire »

Type de données	Stockage	But
binary(n)	1 à 8000 octets	Stocke des données binaires à taille fixe
varbinary(n)	1 à 8000 octets	Stocke des données binaires à taille variable
varbinary(max)	Jusqu'à 2 Go	Stocke des données binaires à taille variable
image	Jusqu'à 2 Go	Stocke des données binaires à taille variable

3.1.2.7 Types de données spécialisés

Type de données	But
bit	Stocke 0, 1 ou null. TRUE est converti en 1 et FALSE en 0.
timestamp	Stocke une valeur automatiquement générée. Chaque base de données contient un compteur interne qui correspond à un minuteur relatif non associé à une horloge réelle. Une table ne peut posséder qu'une colonne timestamp. Celle-ci est fixée à la valeur de l'horodateur de la base de données lors de l'insertion ou de la modification de la ligne.
uniqueidentifier	Contient un identifiant unique (GUID) sur 16 octets qui sert à identifier de façon unique une ligne parmi, par exemples, des bases de données, des instances et des serveurs.
sql_variant	Peut modifier le type de données selon les données stockées dans la colonne. Stocke un maximum de 8000 octets.
xml	Stocke un document XML d'une taille allant jusqu'à 2 Go. Vous pouvez spécifier des options pour n'autoriser que le stockage de documents formés dans la colonne.

D'autres types de données existent. Nous pouvons notamment citer les types spatiaux suivants : geometry et geography.

NB :
Un « well-formed » XML document (document XML bien formé) est un document XML qui a une syntaxe XML correcte. Selon le W3C, cela signifie :

- Les documents XML doivent avoir un élément racine ;
- Les éléments XML doivent avoir une balise de fermeture ;
- Les balises sont « case sensitive » ;
- les éléments XML doivent être positionnés correctement ;
- Les valeurs d'attribut XML doivent toujours être entre guillemets.

Il ne faut pas confondre le well-formed document et le valid XML document, qui se définit comme un « well-formed » XML document qui se conforme aussi aux règles d'un Document Type Definition (DTD) ou d'un XML Schema (XSD), que W3C considère comme une alternative à DTD.

3.1.3 Exemples de création d'une table

Exemple 1

Cette section débute avec un exemple simple. Nous allons utiliser l'instruction CREATE TABLE pour créer les tables contenant les départements d'une université.

```
CREATE TABLE Department (
    Department_ID varchar(20) NOT NULL,
    Department_Name varchar(25)
);
```

Certains aspects de la syntaxe méritent d'être signalés :

- Des parenthèses encadrent la liste des colonnes.
- Dans les colonnes obligatoires, la déclaration NOT NULL suit la spécification du type de données.
- Lorsqu'elle est applicable, la spécification de largeur de colonne apparaît entre parenthèses après celle du type de données.

Exemple 2

```
CREATE TABLE Student (
    Student_ID     varchar(20)  NOT NULL,
    Last_Name      varchar(25)  NOT NULL,
    First_Name     varchar(25)  NOT NULL,
    MI             char(1)    NULL,
    Year           varchar(25)  NULL,
    Street_Address varchar(25)  NULL,
    City           varchar(25)  NULL,
    State          varchar(2)   NULL,
    Zipcode        varchar(9)   NULL,
    Telephone      varchar(10)  NULL,
    Fax            varchar(10)  NULL,
    Email          varchar(100) NULL
);
```

Exemple 3

```
CREATE TABLE CLIENT (
    ID_CLIENT int,
    NOM_F varchar(35),
    RUE varchar(45),
    LOCALITE varchar(25),
    CODE_POST char(6),
    TEL varchar(12)
);
```

Nom de la colonne	Type de données	Autoriser les valeurs Null
ID_CLIENT	int	☑
NOM_F	varchar(35)	☑
RUE	varchar(45)	☑
LOCALITE	varchar(25)	☑
CODE_POST	char(6)	☑
TEL	varchar(12)	☑

3.1.4 Spécification d'une clé primaire

Une clé primaire est un ensemble, formé d'une ou plusieurs colonnes, qui identifie de façon unique chaque ligne de la table.

Voici un autre exemple de syntaxe de l'instruction CREATE TABLE :

```
CREATE TABLE nom_table (
    Nom_colonne1 type_données [NOT NULL],
    …
    nom_colonneN type_données [NOT NULL],
    [Constraint nom_contrainte]
    [Primary key (nom_colonneA, nom_colonneB,    ... nom_colonneX)]
);
```

Les variables sont définies ainsi :

- *nom_table* est le nom de la table.
- *Nom_colonne1* à *nom_colonneN* représentent des noms de colonnes valides.
- *type_données* est la spécification du type de données.
- *nom_contrainte* est un nom optionnel qui identifie la contrainte de clé primaire.
- *nom_colonneA* à *nom_colonneN* représentent les colonnes qui composent la clé primaire.

Bien que la clause de clé primaire dans l'instruction représente un concept essentiel de la théorie du système de base de données relationnel, dans la pratique, son utilisation est optionnelle bien qu'il soit vivement recommandé de créer une clé primaire pour toutes les tables.
Le listing qui suit utilise à nouveau la table Department et montre comment déclarer une clé primaire lors de la création d'une table.

Exemple :

```
CREATE TABLE Department (
    Department_ID varchar(20),
    Department_Name varchar(25),
    Constraint PK_Department PRIMARY KEY (Department_ID)
);
```

Notez que la contrainte NOT NULL n'est pas utilisée pour spécifier que l'attribut Department_ID est une colonne obligatoire. Par défaut, le système contraint les colonnes composant une clé primaire à être obligatoires.

3.1.5 Spécification d'une clé étrangère

Maintenant que vous savez comment spécifier la clé primaire dans l'instruction CREATE TABLE, vous êtes prêt à déclarer vos clés étrangères.

La syntaxe suivante de l'instruction inclut des déclarations de clés primaires et étrangères :

```
CREATE TABLE nom_table (
    spécification_colonne1,
    …
    spécification_colonneN,
    [Constraint nom_contrainte_FK Foreign key (nom_colonneF1, …, nom_co-
    lonneFN)
    references table_référence (nom_colonnePl, …, nom_colonnePN),]
    [Constraint nom_contrainte_PK Primary key (nom_colonneA, nom_colonneB,
    …, nom_colonneX)]
);
```

Les variables sont définies ainsi :

- *nom_table* est le nom de la table.
- *spécification_colonne1* à *spécification_colonneN* sont les spécifications de colonnes valides (décrites ci-après en détail).
- *nom_contrainte_FK* est le nom optionnel de la contrainte de clé étrangère de la table en cours de création.
- *nom_colonneF1* à *nom_colonneFN* représentent les colonnes qui composent la clé étrangère.
- *table_référence* est le nom de la table à laquelle se réfère la déclaration de clé étrangère.
- *nom_colonneP1* à *nom_colonnePN* représentent les colonnes qui composent la clé primaire dans la table de référence.
- *nom_contrainte_PK* est le nom optionnel qui identifie la contrainte de clé primaire de la table en cours de création.
- *nom_colonneA* à *nom_colonneX* représentent les noms de colonnes qui composent la clé primaire.

La syntaxe pour *spécification_colonne* est la suivante :

nom_colonne type_données [DEFAULT *valeur_défaut*]
[CONSTRAINT *nom_contrainte*] [NULL] | [NOT NULL] | [UNIQUE] | CHECK (*condition*)

Les variables sont définies ainsi :

- *nom_colonne* est un nom de colonne valide.
- *type_données* est la spécification d'un type de données valide.
- *valeur_défaut* est une valeur légale assignée par défaut à la colonne lors d'une insertion.
- *nom_contrainte* est un nom valide à assigner à la contrainte.
- *condition* est une condition booléenne valide qui doit être vraie pour qu'une valeur soit assignée à la colonne.

Exemple : création d'une table COURSE

```
CREATE TABLE Course (
   Course_ID       varchar(5),
   Department_ID   varchar(20),
   Title varchar(60) NOT NULL,
   Description varchar(2000),
   Units           tinyint,
   Constraint PK_Course PRIMARY KEY (Course_ID, Department_ID),
   Constraint FK_Course_Department_ID FOREIGN KEY (Department_ID) REFER-
   ENCES Department (Department_ID)
);
```

Rappel :
Une contrainte est un mécanisme qui garantit que les valeurs d'une colonne ou d'un ensemble de colonnes satisfont à une condition déclarée.

Note :
Bien que cela soit optionnel, il est recommandé d'utiliser un nom de contrainte lors de la déclaration d'une clé primaire ou étrangère, ou d'une contrainte de validation. Si vous ne déclarez pas explicitement un nom de contrainte, le SGBD en génère un automatiquement en lui attribuant un nom relativement peu descriptif. Si par la suite vous souhaitez supprimer la clé étrangère, il vous faudra rechercher le nom de contrainte qui aura été généré par le système. Vous vous épargnerez quelques soucis lors de vos recherches en déclarant le nom de la contrainte vous-même. Certains messages d'erreur seront également plus explicites.

Les colonnes citées dans la déclaration de la table de référence doivent composer la clé primaire ou unique de cette table. Si ce n'est pas le cas, le système ne créera pas la clé étrangère. Dans l'exemple suivant, supposons que vous n'ayez pas encore créé la clé étrangère pour la table Course. Tout d'abord, supprimez la clé primaire de la table Department. Lorsque vous essayerez de créer la table Course, le système détectera que l'attribut Department_ID n'est pas défini en tant que clé primaire pour la table Department et l'instruction se conclura en échec.

```
DROP TABLE Course;

ALTER TABLE Department DROP CONSTRAINT PK_Department;

CREATE TABLE Course (
   Course_ID       varchar(5),
   Department_ID   varchar(20),
   Title           varchar(60) NOT NULL,
   Description     varchar(2000),
   Units           tinyint,
   Additional_fees smallmoney,
   Constraint PK_Course PRIMARY KEY (Course_ID, Department_ID),
   Constraint FK_Course_Department_ID FOREIGN KEY (Department_ID) REFER-
   ENCES Department (Department_ID)
);
```

```
Msg 1776, Niveau 16, État 0, Ligne 1
Aucune clé primaire ou candidate dans la table référencée 'Department' ne
correspond à la liste des colonnes de référence de la clé étrangère
'FK_Course_Department_ID'.
Msg 1750, Niveau 16, État 0, Ligne 1
Impossible de créer la contrainte. Voir les erreurs précédentes.
```

3.1.6 Contrainte de valeur avec la clause CHECK

Une puissante fonctionnalité de SQL permet de spécifier, lors de la création de la table, une contrainte qui validera les valeurs insérées dans une colonne. SQL effectue cette tâche à l'aide de la clause optionnelle CHECK qui peut être spécifiée pour chaque colonne. Cette clause permet de définir une condition booléenne dont le résultat ne peut être que vrai (TRUE) ou faux (FALSE). Dans le premier cas, la valeur est acceptée par le système, sinon un code d'erreur est retourné.

Le listing suivant présente un exemple dans lequel la définition de la table Instructor est modifiée afin que la valeur de l'attribut Position soit l'une des trois valeurs autorisées suivantes : 'ASSISTANT PROFESSOR', 'ASSOCIATE PROFESSOR', ou 'PROFESSOR'.

```
CREATE TABLE Instructor (
   Instructor_ID     varchar(20),
   Department_ID     varchar(20)  NOT NULL,
   Last_Name         varchar(25)  NOT NULL,
   First_Name        varchar(25),
   MI                char(1),
   Position          varchar(25) constraint CK_Instructor
   check (Position in ('ASSISTANT PROFESSOR', 'ASSOCIATE PROFESSOR',
                       'PROFESSOR')),
   Telephone         varchar(10),
   Fax               varchar(10),
   Email             varchar(100),
   constraint PK_Instructor Primary Key (Instructor_ID),
   constraint FK_Instructor_Instructor_ID
   Foreign Key (Department_ID) references Department (Department_ID)
);
```

Si on tente d'ajouter un enseignant à la table en lui attribuant un statut (Position) ne faisant pas partie des trois valeurs valides, le système retourne un message d'erreur signalant l'échec d'insertion.

```
INSERT INTO Instructor (Instructor_ID, Department_ID, Last_Name,
First_Name, Position) VALUES('C2222', 'PSYCH', 'MENCKEN', 'H.', 'SECRE-
TARY');

Msg 547, Niveau 16, État 0, Ligne 1
L'instruction INSERT est en conflit avec la contrainte CHECK "CK_Instruc-
tor". Le conflit s'est produit dans la base de données "BD_TEST", table
"dbo.Instructor", column 'Position'.
L'instruction a été arrêtée.
```

Comme 'SECRETARY' n'est pas une valeur valide, la tentative d'insertion de la ligne échoue. Cet exemple a illustré l'utilisation de la contrainte de validation CHECK pour renforcer l'intégrité des données.

3.1.7 Attribution d'une valeur par défaut

La clause DEFAULT, lors de la définition d'une colonne, permet de déclarer une valeur par défaut. Elle sera ensuite utilisée chaque fois qu'une ligne sera insérée à l'aide de l'instruction INSERT, sans spécification de valeur pour une colonne. Par exemple, supposez que la plupart des étudiants habitent dans la ville de Springfield. Vous voulez vous assurer que si un étudiant est ajouté à la table STUDENT sans mention de nom de ville, la valeur « SPRINGFIELD » lui soit attribuée. Le listing ci-dessous illustre comment modifier l'instruction pour obtenir ce résultat.

```
CREATE TABLE Student (
   Student_ID     varchar(20),
   Last_Name      varchar(25)  NOT NULL,
   First_Name     varchar(25)  NOT NULL,
   MI             char(1),
   Year           varchar(25),
   Street_Address varchar(25),
   City           varchar(25) Default 'SPRINGFIELD',
   State          varchar(2),
   Zipcode        varchar(9),
   Telephone      varchar(10),
   Fax            varchar(10),
   Email          varchar(100),
   Constraint PK_Student PRIMARY KEY (Student_ID)
);

INSERT INTO Student(Student_ID, Last_Name, First_Name, Year)
VALUES('109850', 'CARSON', 'RICARDO', 'JUNIOR');

SELECT Student_ID, Last_Name, City
FROM Student
WHERE Student_ID = '109850';

STUDENT_ID           LAST_NAME                CITY
-------------------- ------------------------ -------------
109850               CARSON                   SPRINGFIELD
```

3.2 Modification d'une structure de table avec ALTER TABLE

3.2.1 Introduction

Il est parfois nécessaire de modifier la définition d'une table. L'instruction ALTER TABLE sert à cela. Elle change la structure d'une table, mais pas son contenu. Les types de changements acceptés sont :

- ajout d'une nouvelle colonne dans une table existante;
- augmentation ou diminution de la largeur d'une colonne existante;
- changement de catégorie d'une colonne, d'obligatoire à optionnelle, ou vice versa;
- spécification d'une valeur par défaut pour une colonne existante;
- spécification d'autres contraintes pour une colonne existante.

Voici quelques formes de la syntaxe de base de l'instruction ALTER TABLE :

```
ALTER TABLE nom_table
ADD   spécification_colonne | nom-contrainte , …
      spécification_colonne | nom contrainte;

ALTER TABLE nom_table
ALTER COLUMN spécification_colonne | nom_contrainte;

ALTER TABLE nom_table
DROP CONSTRAINT nom_contrainte;
```

Les variables sont définies ainsi :

- *nom_table* est le nom de la table.
- *spécification_colonne* représente une spécification valide de colonne (nom de colonne et type de données).
- *nom_contrainte* est une contrainte de colonne ou de table.

La première forme de l'instruction est utilisée pour ajouter une colonne, une clé primaire, ou une clé étrangère à une table.
La deuxième forme de l'instruction est utilisée pour modifier une colonne existante. Parmi les actions possibles, vous pouvez augmenter la largeur d'une colonne ou changer sa catégorie, obligatoire ou optionnelle.
La troisième forme de l'instruction ALTER TABLE est utilisée pour supprimer la clé primaire ainsi que d'autres contraintes.

3.2.2 Changement d'une caractéristique de colonne NOT NULL en NULL

Bien que vous puissiez modifier librement une colonne obligatoire pour qu'elle devienne optionnelle, il faut penser aux raisons motivant cette modification. Cette colonne est-elle réellement optionnelle ?

Le listing ci-dessous présente une table nommée Demo_Table. L'instruction ALTER TABLE est utilisée pour modifier la définition de la colonne Current_Value pour qu'elle devienne optionnelle.

- **Créez une table nommée « "demo_table ».**

```
CREATE TABLE demo_table (
    RECORD_NO tinyint NOT NULL,
    DESCRIPTION varchar(40),
    CURRENT_VALUE decimal(15) NOT NULL
);

Name                           Null?
------------------------------ --------
RECORD_NO                      NOT NULL
DESCRIPTION
CURRENT_VALUE                  NOT NULL
```

	Nom de la colonne	Type de données	Autoriser l...
	RECORD_NO	tinyint	☐
	DESCRIPTION	varchar(40)	☑
	CURRENT_VALUE	decimal(15, 0)	☐

- **La commande suivante permet de rendre la colonne Current_Value optionnelle.**

```
ALTER TABLE demo_table ALTER COLUMN current_value decimal(15) null;
```

```
Name                           Null?
------------------------------ --------
RECORD_NO                      NOT NULL
DESCRIPTION
CURRENT_VALUE
```

	Nom de la colonne	Type de données	Autoriser l...
	RECORD_NO	tinyint	☐
	DESCRIPTION	varchar(40)	☑
	CURRENT_VALUE	decimal(15, 0)	☑

3.2.3 Changement d'une caractéristique de colonne NULL en NOT NULL

Si une table est vide, vous pouvez définir une colonne obligatoire avec la mention NOT NULL. Toutefois, si elle ne l'est pas, vous ne pouvez pas transformer une colonne pour qu'elle devienne obligatoire tant que toutes les lignes de la table ne possèdent pas de valeur pour cette colonne.
Le listing suivant montre la réponse du système si vous tentez de transformer la colonne Current_Value pour qu'elle devienne obligatoire. Cependant, si vous vous assurez qu'aucune valeur de ligne pour cette colonne ne possède de valeur nulle, vous pouvez la transformer pour qu'elle devienne obligatoire à l'aide de la déclaration NOT NULL.

Exemple :

- Supposons que la colonne « current_value » contienne des valeurs NULL. La commande suivante se termine par un échec. **ATTENTION : Le message d'erreur renvoyé par le système n'est pas des plus explicites !!!**

```
ALTER TABLE demo_table ALTER COLUMN current_value decimal(15) NOT NULL;
```

```
Msg 515, Niveau 16, État 2, Ligne 1
Impossible d'insérer la valeur NULL dans la colonne 'CURRENT_VALUE',
table 'BD_TEST.dbo.demo_table'. Cette colonne n'accepte pas les valeurs
NULL. Échec de UPDATE.
L'instruction a été arrêtée.
```

- **Ensuite, remplissez la colonne "record_no" de manière arbitraire à l'aide de la commande suivante.**

```
UPDATE demo_table
SET current_value = record_no + 100;
```

- Visualisez ce que la table contient :

```
SELECT * FROM demo_table;
```

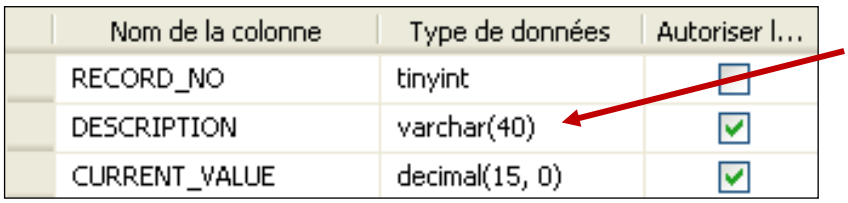
	RECORD_NO	DESCRIPTION	CURRENT_VALUE
1	1	a	101
2	2	b	102
3	3	c	103

- Vous pouvez à présent exécuter la commande permettant d'imposer la contrainte NOT NULL sur la colonne « current_value ».

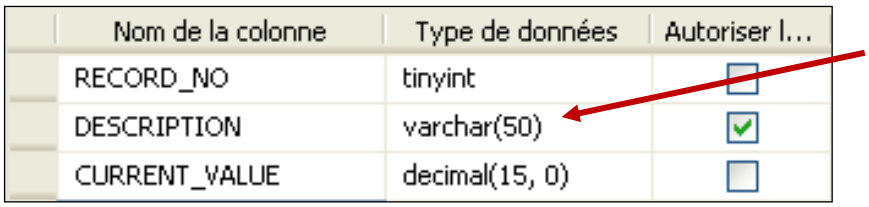
```
ALTER TABLE demo_table ALTER COLUMN current_value decimal(15) NOT NULL;
```

3.2.4 Augmentation d'une largeur de colonne

Vous pouvez utiliser l'instruction ALTER TABLE pour agrandir la largeur d'une colonne. Dans l'exemple ci-dessous, la largeur de 40 caractères de la colonne **Description** est modifiée pour accepter 50 caractères.




```
ALTER TABLE demo_table ALTER COLUMN description varchar(50);
```



3.2.5 Réduction d'une largeur de colonne

Lors de la conception d'une application de base de données, il se peut que vous constatiez qu'une largeur trop grande a été attribuée à une colonne. Vous pourriez considérer cela sans conséquence puisque qu'elle peut recevoir des données et remplir normalement ses fonctions. Au contraire, cela trahit un modèle de données inexact. Pour bien faire, il faut réduire la taille de la colonne afin que sa définition reflète l'espace occupé par les données.

Vous pouvez utiliser l'instruction ALTER TABLE pour réduire la largeur de colonne. Par exemple, supposez que vous souhaitiez employer un caractère unique pour indiquer le nombre d'années d'ancienneté de l'étudiant - 1 pour première année, 2 pour seconde année, etc.

```
ALTER TABLE Student ALTER COLUMN Year varchar(1)
```

Remarques :

- Si vous essayez de réduire la largeur d'un champ et que certaines valeurs ont une longueur qui excède la nouvelle taille définie, vous obtiendrez un message d'erreur semblable à ce qui suit.

  Msg 8152, Niveau 16, État 14, Ligne 1
  Les données de chaîne ou binaires seront tronquées.
  L'instruction a été arrêtée.

- Pour information, avec certains systèmes de gestion de bases de données ne vous autorisent pas à réduire la largeur d'une colonne si elle contient des valeurs. Même si vous mettez à jour la table Student afin que les valeurs de la colonne Year ne dépassent pas 1 caractère, vous ne pourrez toujours pas effectuer la modification. Vous ne pouvez le faire que si la colonne est définie comme NULL.

3.3 Utilisation d'une clé primaire

3.3.1 Introduction

Veillez à définir une clé primaire pour chaque table de la base de données, en voici les raisons.
Une clé primaire empêche que des lignes dupliquées ne soient insérées dans la table. Selon les circonstances, la duplication de lignes peut se révéler problématique et même provoquer le blocage de l'application. Par exemple, vous ne souhaiteriez pas que deux départements aient le même identifiant. Déclarer l'attribut Department_ID comme clé primaire empêcherait que cela ne se produise. Généralement, la clé primaire d'une table est utilisée notamment si une autre table s'y réfère par l'intermédiaire d'une clé étrangère.

3.3.2 Déclaration d'une clé primaire lors de la création d'une table

Vous pouvez définir une clé primaire de table dans l'instruction CREATE TABLE avec la syntaxe suivante :

```
CREATE TABLE nom_table (
    définition_colonne1,
    …
    définition_colonneN,
    [CONSTRAINT nom_contrainte] PRIMARY KEY (nom_colonne1, ...., nom_colonneN)
);
```

Les variables sont définies ainsi :

- *nom_table* est un nom de table valide.
- *définition_colonne1* à *définition_colonneN* représentent des déclarations valides de colonnes.
- *nom_contrainte* est le nom qui est donné à la contrainte de clé primaire.
- *nom_colonne1* à *nom_colonneN* représentent les colonnes qui composent la clé primaire.

Le listing suivant illustre de quelle manière la clé primaire empêche les lignes dupliquées d'être insérées dans la table.

Note : une ligne dupliquée est définie comme étant une ligne dont les valeurs des colonnes qui forment la clé primaire sont identiques à celles des mêmes colonnes d'une autre ligne.

```
INSERT INTO Department(Department_ID, Department_Name) VALUES('PSYCH', 'PSYCHOLOGY');

INSERT INTO Department(Department_ID, Department_Name) VALUES('PSYCH', 'PSYCHIATRY');

Msg 2627, Niveau 14, État 1, Ligne 5
```

```
Violation de la contrainte PRIMARY KEY 'PK_Department'. Impossible d'insé-
rer une clé en double dans l'objet 'dbo.Department'.
L'instruction a été arrêtée.
```

Il est aussi possible de définir la clé comme primaire après que la table a été créée.

Remarques :
Il est parfois nécessaire de connaître le nom d'une contrainte. Par exemple, lorsque vous souhaitez supprimer une clé étrangère, il faut supprimer en même temps la contrainte qui lui est associée. À moins que vous n'ayez personnellement spécifié le nom de la contrainte, vous aurez à rechercher celui attribué par le système. Par conséquent, il est préférable de toujours attribuer un nom de contrainte lors de la déclaration des clés primaires ou étrangères.

Attribuez un nom de contrainte de clé primaire ou de clé étrangère significatif, comme par exemple, *PK_nomdetable* (pour Primary Key) ou *FK_nomdetable_colonne* (pour Foreign Key), en respectant les limites autorisées par le système pour les noms d'objets. Idéalement, il est également conseillé d'attribuer des noms aux contraintes de validation ou d'unicité, par exemple, *nomdetable_colonne_CK* (pour Check) ou *nomdetable_colonne_UN* (pour Unique).

3.3.3 Déclaration d'une clé primaire sur une table existante

Vous avez la possibilité de définir une contrainte de clé primaire après qu'une table ait été créée. En effet, une table peut ne pas avoir de clé primaire déjà définie. Pour en déclarer une sur une table existante, utilisez l'instruction ALTER TABLE.

```
CREATE TABLE Department (
    Department_ID varchar(20) NOT NULL,
    Department_Name varchar(25) CONSTRAINT NN_Department_Name NOT NULL
);

ALTER TABLE Department ADD CONSTRAINT PK_Department primary key (Department_ID);
```

NB : Il faut que le(s) champ(s) constitutif(s) de la clé primaire soit(soient) de type NOT NULL sinon la commande se termine par un échec.
Une contrainte de clé primaire est appliquée dès qu'elle a été définie, lors de la création de la table ou ultérieurement.

3.4 Clé étrangère et intégrité référentielle

3.4.1 Introduction

Les clés primaires et étrangères fonctionnent de concert pour appliquer la contrainte d'intégrité référentielle. Une clé étrangère dans une table est une colonne, ou un ensemble de colonnes, dont les valeurs sont limitées à celles de la clé primaire d'une autre table. Définissez des clés étrangères lorsque cela est possible. Pour une application client-serveur, la première ligne de défense de l'intégrité référentielle est le programme applicatif client, et la dernière, les clés primaires et étrangères définies dans la base de données.

Comme une clé primaire, une clé étrangère peut être déclarée après la création de la table.

3.4.2 Déclaration d'une clé étrangère lors de la création d'une table

Pour illustrer la définition d'une clé étrangère, analysons les tables Department et Instructor de notre exemple de base de données. Chaque département possède un identifiant unique enregistré dans la colonne Department_ID. Un enseignant doit être associé à un seul département, même si, dans une application réelle, il n'est pas impossible qu'il puisse faire partie de plusieurs.

Le système assure l'intégrité référentielle en interdisant toute opération qui attribuerait une valeur de clé étrangère n'existant pas comme clé primaire dans la table à laquelle la clé étrangère se réfère. Par exemple, le listing suivant montre que vous ne pouvez pas insérer une ligne précisant qu'un enseignant fait partie d'un département identifié par « PHYSICS », car cette valeur n'existe pas dans l'attribut Department_ID de la table Department.

```
INSERT INTO Instructor(Instructor_ID, Department_ID, Last_Name)
VALUES('S1000', 'PHYSICS', 'MICHELSON');

Msg 547, Niveau 16, État 0, Ligne 1
L'instruction INSERT est en conflit avec la contrainte FOREIGN KEY "FK_Ins-
tructor_Department". Le conflit s'est produit dans la base de données
"BD_TEST", table "dbo.Department", column 'Department_ID'.
L'instruction a été arrêtée.
```

Voici un exemple de création d'une clé étrangère dans une table.

```
CREATE TABLE Course (
    Course_ID varchar(5),
    Department_ID varchar(20) CONSTRAINT NN_Course_Department_ID NOT NULL,
    Title varchar(60) NOT NULL,
    Description varchar(2000),
    Units tinyint,
    Additional_fees smallmoney,
    CONSTRAINT PK_Course PRIMARY KEY (Course_ID, Department_ID),
    CONSTRAINT FK_Course_Department_ID FOREIGN KEY (Department_ID)
    REFERENCES Department (Department_ID)
);
```

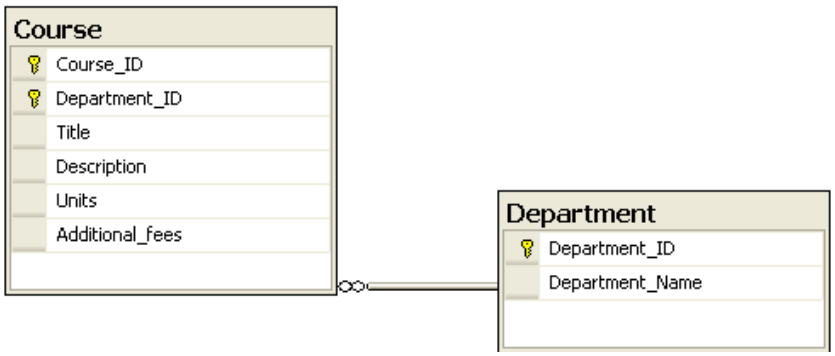
Nom de la colonne	Type de données	Autoriser les valeurs Null
Course_ID	varchar(5)	☐
Department_ID	varchar(20)	☐
Title	varchar(60)	☐
Description	varchar(2000)	☑
Units	tinyint	☑
Additional_fees	smallmoney	☑

3.4.3 Déclaration d'une clé étrangère sur une table existante

Il est possible de déclarer une clé étrangère après qu'une table ait été créée. C'est l'instruction ALTER TABLE qui sera une nouvelle fois utilisée.

```
CREATE TABLE Course (
    Course_ID varchar(5),
    Department_ID varchar(20) CONSTRAINT NN_Course_Department_ID NOT NULL,
    Title varchar(60) CONSTRAINT NN_Course_Title NOT NULL,
    Description varchar(2000),
    Units tinyint,
    Additional_fees smallmoney,
    CONSTRAINT PK_Course PRIMARY KEY (Course_ID, Department_ID)
);
```

```
ALTER TABLE Course add CONSTRAINT FK_Course_Department_IDFOREIGN KEY
(Department_ID) REFERENCES Department (Department_ID);
```



3.4.4 Vérification des relations entre clés primaires et étrangères

Lorsque vous définissez une clé étrangère, le système effectue les vérifications suivantes :

- qu'une clé primaire ait été définie pour la table à laquelle se réfère la clé étrangère;
- que le nombre de colonnes composant la clé étrangère corresponde à celui de la clé primaire;
- que la définition du type et de la taille de chaque colonne de la clé étrangère soit identique à celle de la colonne correspondante dans la clé primaire.

Le listing suivant illustre ce qui se produit lorsque le type d'une colonne de clé primaire ne correspond pas à celui de la colonne prévue comme clé étrangère. Par exemple, la table Employee contient des employés qui travaillent pour une société et la table Employee_Dependent contient les personnes à la charge de ces employés. La clé primaire de la table Employee est Employee_ID. Bien sûr, la table Employee_Dependent contient aussi l'attribut Employee_ID. Notez que la définition d'Employee_ID de la table Employee est DECIMAL(4), mais que la définition de cette colonne dans la table Employee_Dependent est VARCHAR(4). Par conséquent, le système n'autorise pas que cette colonne soit définie comme clé étrangère dans la table Employee_Dependent si elle se réfère à la colonne de la table Employee.

Employee

```
Name                           Null?    Type
------------------------------ -------- ----
EMPLOYEE_ID                    NOT NULL DECIMAL(4)
LAST_NAME                      NOT NULL VARCHAR(30)
FIRST_NAME                     NOT NULL VARCHAR(20)
MIDDLE_INITIAL                          CHAR(1)
HIRE_DATE                      NOT NULL DATETIME
TERMINATION_DATE                        DATETIME
DATE_OF_BIRTH                           DATETIME
MONTHLY_SALARY                          DECIMAL(5)
MANAGER                        NOT NULL CHAR(1)
USERNAME                                VARCHAR(31)
```

Employee_Dependent

```
Name                           Null?    Type
------------------------------ -------- ----
EMPLOYEE_ID                    NOT NULL VARCHAR(4)
LAST_NAME                               VARCHAR(30)
FIRST_NAME                              VARCHAR(20)
MIDDLE_INITIAL                          CHAR(1)
RELATIONSHIP                            VARCHAR(30)
```

```
ALTER TABLE Employee_Dependent ADD CONSTRAINT Employee_Dependent_FK1 FOREIGN KEY
(Department_ID) REFERENCES Employee(Employee_ID);

Msg 1778, Niveau 16, État 0, Ligne 1
La colonne 'Employee.Employee_id' n'appartient pas au même type de données
que la colonne de référence 'Employee_Dependent.Employee_id' dans la clé
étrangère 'Employee_Dependent_FK1'.
Msg 1750, Niveau 16, État 0, Ligne 1
Impossible de créer la contrainte. Voir les erreurs précédentes.
```

3.4.5 Activation ou désactivation des contraintes d'intégrité

Comme mentionné précédemment, les contraintes de clés primaires et étrangères permettent de mettre en application deux concepts essentiels du modèle relationnel: l'intégrité des données et l'intégrité référentielle. Il est possible que vous souhaitiez désactiver ces contraintes, comme durant la conception d'une base ou la migration de données existantes d'une organisation.

3.4.6 Suppression d'une clé primaire

Durant la conception d'une base de données, il peut être utile de supprimer la clé primaire d'une table.

Pour cela, utilisez l'instruction ALTER TABLE avec la syntaxe suivante :

```
ALTER TABLE nom_table DROP CONSTRAINT nom_contrainte_pk;
```

La variable est définie ainsi :

- ✓ *nom_table* représente la table contenant la clé primaire.
- ✓ *Nom_contrainte_pk* représente la contrainte de type clé primaire.

À titre d'illustration, supposez que vous conceviez une table AP_Header pour stocker des informations concernant les créances à payer et déclariez une clé primaire sur la colonne Vendor_Invoice_Number, « numéro de facture fournisseur » (voir listing ci-après). Après avoir inspecté certaines données et réfléchi à la définition de la table, vous réalisez que deux fournisseurs différents pourraient parfaitement utiliser le même numéro de facture. Par conséquent, vous supprimez la clé primaire de la table.

```
CREATE TABLE AP_Header(
    Bill_Number decimal(4) NOT NULL,
    Vendor_Invoice_Number varchar(10),
    Vendor_ID varchar(6)  NOT NULL,
    Date_Received datetime NOT NULL,
    Bill_Status varchar(5),
    CONSTRAINT PK_Ap_Header PRIMARY KEY (Vendor_Invoice_Number)
);

ALTER TABLE AP_Header drop constraint PK_Ap_Header;
```

Qu'en serait-il s'il existait une clé étrangère dans la table des détails de créances AP_Detail pointant sur la clé primaire de la table principale AP_Header ? Si c'était le cas, le système ne permettrait pas de supprimer la clé primaire dans cette dernière. Le code qui suit illustre cette situation.

```
CREATE TABLE AP_Detail (
    Bill_Number            decimal(4) NOT NULL,
    Vendor_Invoice_Number  varchar(10) NOT NULL,
    Item_Number            decimal(3) NOT NULL,
    Billed_Amount          decimal(8,2) NOT NULL,
```

```
    Approved_Amount         decimal(8,2),
    Paid_Amount             decimal(8,2),
    CONSTRAINT AP_Detail_FK FOREIGN KEY (Vendor_Invoice_Number) REFERENCES AP_Header
);
```

```
ALTER TABLE AP_Header drop constraint PK_Ap_Header;
```

```
Msg 3725, Niveau 16, État 0, Ligne 1
La contrainte 'PK_Ap_Header' est actuellement référencée par la table
'AP_Detail', contrainte de clé étrangère 'AP_Detail_FK'.
Msg 3727, Niveau 16, État 0, Ligne 1
Impossible de supprimer la contrainte. Consultez les erreurs précédentes.
```

Note

Avec Oracle par exemple, il existe une option CASCADE pour la clause DROP PRIMARY KEY. Utilisez-la avec prudence, car elle permet de supprimer la clé primaire ainsi que les clés étrangères qui s'y réfèrent. Le listing suivant illustre la suppression d'une clé primaire avec cette option sous Oracle. Cette option n'est pas disponible avec SQL Server.

```
ALTER TABLE AP_Header DROP PRIMARY KEY CASCADE;
```

3.4.7 Suppression d'une clé étrangère

Durant la conception de la base de données, il se peut que vous réalisiez avoir défini par erreur une colonne en tant que clé étrangère. Il faut alors que vous indiquiez le nom de contrainte associé à la clé étrangère en utilisant la syntaxe suivante :

```
ALTER TABLE nom_table DROP CONSTRAINT nom_contrainte
```

Les variables sont définies ainsi :

- *nom_table* est le nom de la table contenant la clé étrangère.
- *nom_contrainte* est le nom de contrainte associé à la clé étrangère.

Supposez que la table AP_Header possède une colonne supplémentaire nommée Vendor_Status qui, par coïncidence, possède le même type de données et la même taille que la colonne Vendor_ID. Comme le montre le listing suivant, vous déclarez par erreur une clé étrangère sur Vendor_Status qui pointe sur la clé primaire d'une table Vendor. Après avoir tenté d'insérer une valeur dans la colonne Vendor_Status, vous réalisez votre erreur et supprimez la clé étrangère assignée à la colonne.

```
ALTER TABLE AP_Header ADD CONSTRAINT AP_Header_Vendor_Status_FK FOREIGN
KEY (Vendor_Status) REFERENCES Vendor;
```

```
ALTER TABLE AP_Header DROP CONSTRAINT AP_Header_Vendor_Status_FK;
```

3.5 Déclaration d'une contrainte d'unicité

3.5.1 Présentation

Outre la déclaration de clés primaires et étrangères, le système permet de déclarer une colonne pour qu'elle ne reçoive que des valeurs uniques. Une contrainte d'unicité ne se substitue pas à une contrainte de clé primaire. Imaginez une table « Patient » contenant une liste des patients d'un hôpital. Chaque patient reçoit un numéro d'identification contenu dans la colonne Patient_ID qui est utilisée comme clé primaire. Toutefois, chaque patient possède aussi un numéro de sécurité sociale qui est unique. Le listing suivant montre un exemple de mise en application de la contrainte d'unicité pour garantir l'unicité des numéros de sécurité sociale.

```
CREATE TABLE Patient (
    Patient_ID varchar(6) PRIMARY KEY,
    Last_Name varchar(30) NOT NULL,
    First_Name varchar(20) NOT NULL,
    Middle_Name varchar(20),
    Social_Security_Number varchar(9) UNIQUE,
    Insurance_Carrier_Code varchar(4)
);

INSERT INTO Patient(Patient_ID, Last_Name, First_Name) VALUES('A901',
'NORTON', 'ED');

INSERT INTO Patient(Patient_ID, Last_Name, First_Name, Social_Security_Number)
VALUES('A902', 'KRAMDEN', 'RALPH', '123456789');

INSERT INTO Patient(Patient_ID, Last_Name, First_Name, Social_Security_Number)
VALUES('A903','NORTON', 'TRIXIE', '123456789');

Msg 2627, Niveau 14, État 1, Ligne 1
Violation de la contrainte UNIQUE KEY 'UQ__Patient__0F194668412EB0B6'. Im-
possible d'insérer une clé en double dans l'objet 'dbo.Patient'.
L'instruction a été arrêtée.
```

3.5.2 Différences entre clés primaires et contraintes d'unicité

Il existe deux différences entre une clé primaire et une contrainte d'unicité qui méritent d'être soulignées. La première, est qu'il ne peut y avoir qu'une seule clé primaire dans une table, alors qu'il est possible d'avoir plusieurs contraintes d'unicité. La deuxième, est que les colonnes composant une clé primaire sont rendues obligatoires de façon mécanique, alors que celles faisant l'objet d'une contrainte d'unicité ne le sont pas. Si vous souhaitez que les colonnes soumises à une contrainte d'unicité soient obligatoires, il faut ajouter une spécification NOT NULL.

3.6 Index de table

3.6.1 Présentation

Toute discussion relative aux contraintes de clés primaires et étrangères doit inclure les index, ces éléments étant étroitement liés. Cette section décrit les index et leur utilisation par le développeur et le système.

Le système fournit deux types d'index: unique et non unique. Les premiers appliquent notamment les contraintes de clé primaire et d'unicité. Les seconds peuvent améliorer les performances des requêtes.

Notes :

- Une clause ORDER BY d'une instruction SELECT peut se référer à n'importe quelle colonne d'une table, qu'elle possède ou non un index existant sur cette colonne.
- Généralement les index ne peuvent pas être composés de plus de 16 colonnes. Cette valeur est susceptible de varier suivant les SGBD.
- Un index ne stocke pas les valeurs NULL.
- Généralement, on ne dispose pas d'instruction permettant d'examiner le contenu d'un index.

À ne pas faire :

- Créer un index unique au lieu de déclarer une clé primaire.
- Créer un nombre d'index excessif sur une table. Vous pourriez améliorer la recherche des données, mais aussi affecter de façon négative les performances globales de la base de données.
- Créer un index sur des petites tables.

3.6.2 Création d'un index

Voici la syntaxe de base utilisée pour créer un index:

```
CREATE [UNIQUE] INDEX nom_index
ON nom_table (colonne1, ... colonneN);
```

Les variables sont définies ainsi :

- *nom_index* est le nom à donner à l'index. Il est sujet aux limitations d'appellation d'objet de base de données du système concerné.
- *nom_table* est la table pour laquelle l'index est créé.
- *colonne1* à *colonneN* représentent les colonnes utilisées pour créer l'index.

```
CREATE INDEX Student_Last_Name
ON Student(Last_Name);
```

3.6.3 Suppression d'un index

Syntaxe :

```
DROP INDEX nom_table.nom_index;
```

Exemple : `DROP INDEX Student.Student_Last_Name;`

3.6.4 Réserves sur la création d'un index UNIQUE

Bien que CREATE UNIQUE INDEX soit une instruction SQL valide, vous devriez éviter de l'utiliser. Déclarez plutôt des contraintes PRIMARY KEY et UNIQUE. Les deux raisons principales sont les suivantes :

- Lorsque vous déclarez des contraintes PRIMARY KEY et UNIQUE, le système crée automatiquement l'index UNIQUE approprié pour les appliquer.
- Vous ne pourrez pas déclarer de contrainte FOREIGN KEY à moins que vous ne déclariez aussi une contrainte PRIMARY KEY (ou UNIQUE) correspondante.

3.7 Visualisation des contraintes

3.7.1 Utilisation de DESCRIBE

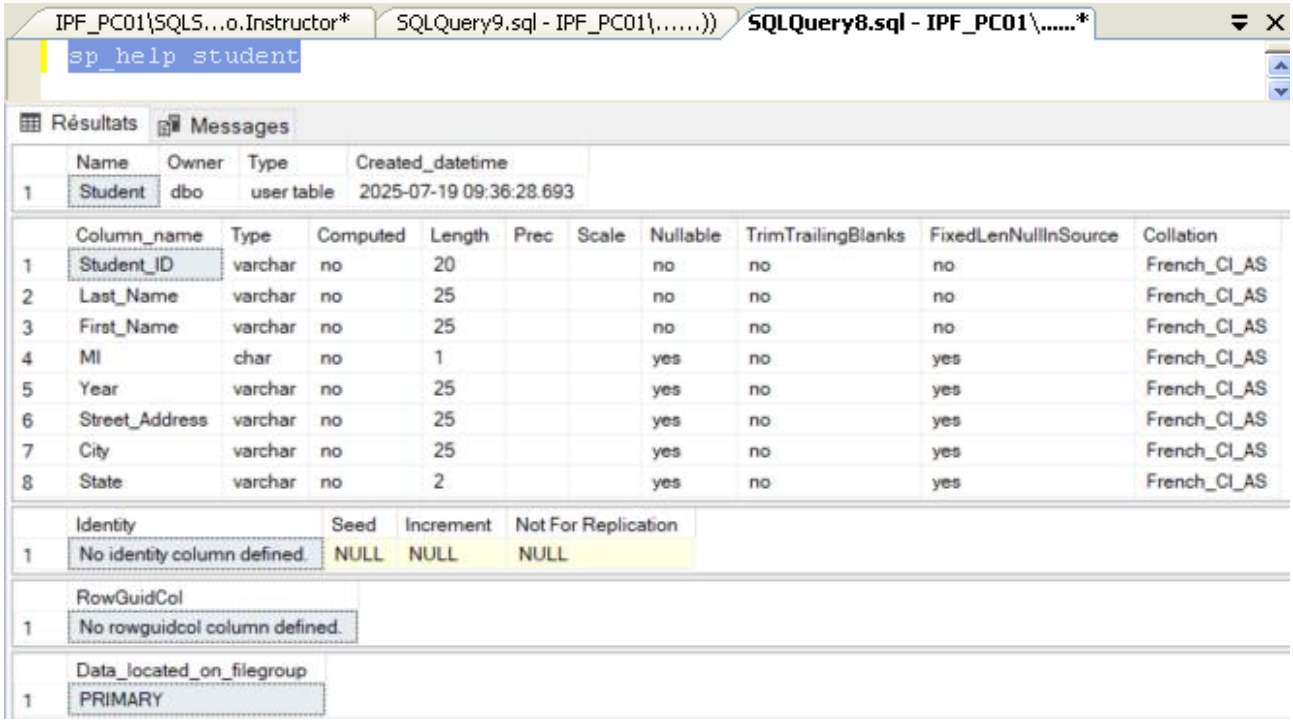
Oracle par exemple, offre la possiblité d'utiliser la commande DESCRIBE. De plus, lorsque vous déclarez une colonne NOT NULL, le système gère l'obligation de valeur comme une contrainte. En fait, cette contrainte est la seule qui peut être mise en évidence à l'aide de la commande SQL*PLUS DESCRIBE (voir listing ci-dessous). Si vous examinez la table **Instructor** dans SQL*Plus, le système indique que cette table possède trois colonnes obligatoires. Il faut noter que la colonne Instructor_ID a été définie comme clé primaire de la table et est donc automatiquement NOT NULL.

```
DESCRIBE Instructor

Name                           Null?    Type
------------------------------ -------- ----
INSTRUCTOR_ID                  NOT NULL varchar(20)
DEPARTMENT_ID                  NOT NULL varchar(20)
LAST_NAME                      NOT NULL varchar(25)
FIRST_NAME                              varchar(25)
MI                                      varchar(1)
POSITION                                varchar(25)
TELEPHONE                               varchar(10)
FAX                                     varchar(10)
EMAIL                                   varchar(100)
```

L'instruction DESCRIBE n'est pas disponible avec SQL Server 2008. Avec ce S.G.B.D., nous utiliserons la procédure stockée système *sp_help*.

Exemple :



...

3.7.2 Utilisation de vues systèmes afin de visualiser les contraintes

Exemple 1 :

```
SELECT OBJECT_NAME(object_id) as NomContrainte,
           SCHEMA_NAME(schema_id) AS NomSchema,
           OBJECT_NAME(parent_object_id) AS NomTable,
           type_desc as TypeContrainte
FROM sys.objects
WHERE type_desc LIKE '%CONSTRAINT'
```

Résultats | Messages

	NomContrainte	NomSchema	NomTable	TypeContrainte
1	PK__sysdiagr__C2B05B61023D5A04	dbo	sysdiagrams	PRIMARY_KEY_CONSTRAINT
2	UK_principal_name	dbo	sysdiagrams	UNIQUE_CONSTRAINT
3	PK_Instructor	dbo	Instructor	PRIMARY_KEY_CONSTRAINT
4	CK_Instructor	dbo	Instructor	CHECK_CONSTRAINT
5	PK_Student	dbo	Student	PRIMARY_KEY_CONSTRAINT
6	DF__Student__City__276EDEB3	dbo	Student	DEFAULT_CONSTRAINT
7	PK_Department	dbo	Department	PRIMARY_KEY_CONSTRAINT
8	FK_Instructor_Department	dbo	Instructor	FOREIGN_KEY_CONSTRAINT
9	PK_Course	dbo	Course	PRIMARY_KEY_CONSTRAINT
10	FK_Course_Department_ID	dbo	Course	FOREIGN_KEY_CONSTRAINT
11	PK_Ap_Header	dbo	AP_Header	PRIMARY_KEY_CONSTRAINT
12	PK__Patient__C1A88B593E52440B	dbo	Patient	PRIMARY_KEY_CONSTRAINT
13	UQ__Patient__0F194668412EB0B6	dbo	Patient	UNIQUE_CONSTRAINT
14	PK_Table_1	dbo	Table_1	PRIMARY_KEY_CONSTRAINT

Exemple 2 :

```
SELECT *
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
```

Résultats | Messages

	CONSTRAINT_CATALOG	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	CONSTRAINT_TYPE
1	BD_TEST	dbo	PK__sysdiagr__C2B05B61023D5A04	BD_TEST	dbo	sysdiagrams	PRIMARY KEY
2	BD_TEST	dbo	UK_principal_name	BD_TEST	dbo	sysdiagrams	UNIQUE
3	BD_TEST	dbo	PK_Instructor	BD_TEST	dbo	Instructor	PRIMARY KEY
4	BD_TEST	dbo	CK_Instructor	BD_TEST	dbo	Instructor	CHECK
5	BD_TEST	dbo	PK_Student	BD_TEST	dbo	Student	PRIMARY KEY
6	BD_TEST	dbo	PK_Department	BD_TEST	dbo	Department	PRIMARY KEY
7	BD_TEST	dbo	FK_Instructor_Department	BD_TEST	dbo	Instructor	FOREIGN KEY
8	BD_TEST	dbo	PK_Course	BD_TEST	dbo	Course	PRIMARY KEY
9	BD_TEST	dbo	FK_Course_Department_ID	BD_TEST	dbo	Course	FOREIGN KEY
10	BD_TEST	dbo	PK_Ap_Header	BD_TEST	dbo	AP_Header	PRIMARY KEY
11	BD_TEST	dbo	PK__Patient__C1A88B593E52440B	BD_TEST	dbo	Patient	PRIMARY KEY
12	BD_TEST	dbo	UQ__Patient__0F194668412EB0B6	BD_TEST	dbo	Patient	UNIQUE
13	BD_TEST	dbo	PK_Table_1	BD_TEST	dbo	Table_1	PRIMARY KEY

Exemple 3 :

```
SELECT RO.NAME AS ParentTable, RC.NAME AS ParentColumn, FO.NAME AS ForeignTable, FC.NAME AS ForeignColumn
FROM sysforeignkeys F INNER JOIN sysobjects RO
ON F.rkeyid = RO.id INNER JOIN syscolumns RC
ON RC.id = RO.id AND RC.colid = F.rkey INNER JOIN sysobjects FO
ON F.fkeyid = FO.id INNER JOIN syscolumns FC
ON FC.id = FO.id AND FC.colid = F.fkey
ORDER BY RO.NAME, RC.NAME, FO.NAME, FC.NAME
```

Résultats | Messages

	ParentTable	ParentColumn	ForeignTable	ForeignColumn
1	AP_Header	Vendor_Invoice_Number	AP_Detail	Vendor_Invoice_Number
2	Class	Class_ID	Student_Schedule	Class_ID
3	Schedule_Type	Schedule_ID	Schedule_Type_Details	Schedule_ID
4	Student	Student_ID	Student_Schedule	Student_ID

3.8 Limitation de valeurs avec la contrainte CHECK

3.8.1 Présentation

La contrainte de validation CHECK est une contrainte de niveau colonne qui peut servir deux objectifs :

- Certaines colonnes, souvent numériques, qui ne possèdent pas un ensemble de valeurs distinctes pouvant être référencées par une clé étrangère, n'acceptent parfois qu'une certaine plage de valeurs prédéfinie. Le contrôle pourra être fait à l'aide d'une contrainte CHECK.
- Lorsque la plage des valeurs autorisées pour une colonne n'est pas constante, et que les valeurs doivent être calculées au moment de l'insertion ou de la mise à jour, l'application de la formule pourra être mise en place avec une contrainte CHECK.

La contrainte CHECK est déclarée dans une instruction CREATE TABLE ou ALTER TABLE selon la syntaxe suivante :

nom_colonne type_données [CONSTRAINT *nom_contrainte*] [CHECK (*condition*)]

Les variables sont définies ainsi :

- ✓ *nom_colonne* est le nom de la colonne.
- ✓ *type_données* est le type de la colonne, avec, le cas échéant, la largeur et l'échelle.
- ✓ *nom_contrainte* est le nom de la contrainte.
- ✓ *condition* est une condition valide retournant une valeur booléenne.

Pour illustrer ce concept, le listing suivant montre comment vous pouvez créer une table de base de données utilisée dans un hôpital pour conserver des informations sur les patients. Une des colonnes de la table concerne la température du corps en degrés Celsius. En la déclarant de type NUMBER (3,1) les valeurs possibles se situent entre 0,0 et 99,9 incluant encore des valeurs absurdes pour exprimer une température du corps. Il est alors possible de recourir à une contrainte de validation CHECK pour limiter les entrées aux valeurs situées entre 25,0 (pour les patients souffrant d'hypothermie (très) grave) et 43,0.

```
CREATE TABLE Patient (
   Patient_ID     varchar(6) PRIMARY KEY,
   Body_Temp_Deg_C  decimal(4,1) constraint Patient_Body_Temp
   CHECK (Body_Temp_Deg_C >= 25.0 and Body_Temp_Deg_C <= 43.0)
);
```

Autre syntaxe possible :

```
CREATE TABLE Patient (
   Patient_ID     varchar(6) PRIMARY KEY,
   Body_Temp_Deg_C  decimal(4,1) constraint Patient_Body_Temp
   CHECK (Body_Temp_Deg_C between 25.0 and 43.0)
);

INSERT INTO Patient(Patient_ID, Body_Temp_Deg_C) VALUES('A1001', 37.6);
```

L'enregistrement est créé sans problème.

```
INSERT INTO Patient(Patient_ID, Body_Temp_Deg_C) VALUES('Q7777', 51.7);

Msg 547, Niveau 16, État 0, Ligne 1
L'instruction INSERT est en conflit avec la contrainte CHECK "Patient_Body_Temp". Le conflit s'est produit dans la base de données "BD_TEST", table "dbo.Patient", column 'Body_Temp_Deg_C'.
L'instruction a été arrêtée.
```

Vous pouvez utiliser des fonctions SQL intégrées ainsi que différents opérateurs dans une contrainte CHECK. Le listing suivant donne un exemple de code permettant de vérifier que le statut d'assuré d'un patient est bien 'Y' ou 'N'. On utilise alors l'opérateur IN dans la contrainte CHECK.

```
CREATE TABLE Patient (
   Patient_ID     varchar(6) PRIMARY KEY,
   Body_Temp_Deg_C  decimal(3,1) CONSTRAINT Patient_Body_Temp
   CHECK (Body_Temp_Deg_C >= 25.0 and Body_Temp_Deg_C <= 43.0),
   Insurance_Status Char(1) CONSTRAINT Patient_Insurance_Status
   CHECK (Insurance_Status in ('Y','y','N','n'))
);

INSERT INTO Patient(Patient_ID, Insurance_Status) VALUES('R4321','Y');
```

L'enregistrement est créé sans problème.

```
INSERT INTO Patient(Patient_ID, Insurance_Status) VALUES('U3030','U');

Msg 547, Niveau 16, État 0, Ligne 1
L'instruction INSERT est en conflit avec la contrainte CHECK "Patient_Insurance_Status". Le conflit s'est produit dans la base de données "BD_TEST", table "dbo.Patient", column 'Insurance_Status'.
L'instruction a été arrêtée.
```

Comme vous le voyez, un message signalant la violation de la contrainte est affiché lorsque le programme tente d'insérer la valeur 'U' dans la colonne Insurance_Status.

3.8.2 Contraintes CHECK multiples sur une colonne

Le système ne limite pas le nombre de contraintes de validation qui peuvent être définies sur une colonne ou une table. Le listing qui suit illustre l'exemple d'une colonne, Amount_Approved, pour laquelle deux contraintes CHECK ont été définies.

```
CREATE TABLE Loan_Application (
   Loan_Application_No      decimal(6) PRIMARY KEY,
   Borrower_Last_Name       varchar(30) NOT NULL,
   Borrower_First_Name      varchar(20) NOT NULL,
   Borrower_Middle_Name     varchar(20),
   Amount_Requested         decimal (9,2) NOT NULL,
   Amount_Approved          decimal (9,2)
                            CONSTRAINT Amount_Approved_Limit
                            CHECK (Amount_Approved <= 1000000),
                            CONSTRAINT Amount_Approved_Interval
                            CHECK ((Amount_Approved % 1000) = 0)
);

INSERT INTO Loan_Application(Loan_Application_No, Borrower_Last_Name,
Borrower_First_Name, Amount_Requested, Amount_Approved)
VALUES(2001, 'RUBRIK', 'STANLEY', 1000000, 999950);

Msg 547, Niveau 16, État 0, Ligne 1
L'instruction INSERT est en conflit avec la contrainte CHECK "Amount_Approved_Interval". Le conflit s'est produit dans la base de données "BD_TEST", table "dbo.Loan_Application", column 'Amount_Approved'.
L'instruction a été arrêtée.


INSERT INTO Loan_Application(Loan_Application_No, Borrower_Last_Name,
Borrower_First_Name, Amount_Requested, Amount_Approved) VALUES(2001,
'RUBRIK', 'STANLEY', 1000000, 999000);
```

L'enregistrement est créé sans problème.

Dans l'exemple du listing précédent, deux contraintes sont définies pour la table Loan_Application. La première limite le montant approuvé à 1 million de dollars, et la seconde garantit que le montant approuvé est donné par incréments de mille dollars. Vous pouvez aisément combiner deux contraintes en une seule. Toutefois, il est préférable de les séparer au cas où l'une d'elles devrait être désactivée.

3.8.3 Référence à d'autres colonnes dans une contrainte CHECK

Une des limitations de l'emploi d'une contrainte CHECK de colonne est qu'elle ne peut pas se référer à une autre colonne de la table sur laquelle elle est définie. Néanmoins il est possible d'utiliser une contrainte CHECK de niveau table pour rendre cela possible. Imaginez que vous soyez responsable de la création d'une table prévue pour stocker des informations relatives à des prêts. Dans cette table, la colonne Amount_Requested doit contenir le montant demandé par l'emprunteur, et la colonne Amount_Approved, le montant accordé par le comité d'approbation. Une des règles de gestion stipule que le prêteur ne doit jamais accorder un montant supérieur à celui qui a été sollicité.

Le listing suivant illustre ce scénario. Pour spécifier une contrainte CHECK de table, il suffit d'ajouter une virgule à la suite de la définition de la colonne concernée, avant la contrainte. Dans notre exemple, il s'agit de la colonne Amount_Approved.

```
CREATE TABLE Loan_Application (
    Loan_Application_No      decimal(6) PRIMARY KEY,
    Borrower_Last_Name       varchar(30) NOT NULL,
    Borrower_First_Name      varchar(20) NOT NULL,
    Borrower_Middle_Name     varchar(20),
    Amount_Requested         decimal(9,2) NOT NULL,
    Amount_Approved          decimal(9,2)
                             CONSTRAINT Amount_Approved_Limit
                             CHECK (Amount_Approved <= Amount_Requested)
);

Msg 8141, Niveau 16, État 0, Ligne 1
La contrainte CHECK de la colonne 'Amount_Approved' référence une autre
colonne, table 'Loan_Application'.
Msg 1750, Niveau 16, État 0, Ligne 1
Impossible de créer la contrainte. Voir les erreurs précédentes.


CREATE TABLE Loan_Application(
    Loan_Application_No      decimal (6) PRIMARY KEY,
    Borrower_Last_Name       varchar(30) NOT NULL,
    Borrower_First_Name      varchar(20) NOT NULL,
    Borrower_Middle_Name     varchar(20),
    Amount_Requested         decimal (9,2) NOT NULL,
    Amount_Approved          decimal (9,2),
    CONSTRAINT Amount_Approved_Limit
    CHECK (Amount_Approved <= Amount_Requested)
);
```

Une virgule qui a toute son importance.

La table est créée sans problème.

```
INSERT INTO Loan_Application(Loan_Application_No, Borrower_Last_Name,
Borrower_First_Name, Amount_Requested, Amount_Approved) VALUES(2001,
'CRANDALL', 'JULIE', 300000, 310000);

Msg 547, Niveau 16, État 0, Ligne 1
L'instruction INSERT est en conflit avec la contrainte CHECK "Amount_Ap-
proved_Limit". Le conflit s'est produit dans la base de données "BD_TEST",
table "dbo.Loan_Application".
L'instruction a été arrêtée.


INSERT INTO Loan_Application(Loan_Application_No, Borrower_Last_Name,
Borrower_First_Name, Amount_Requested, Amount_Approved) VALUES(2001,
'CRANDALL', 'JULIE', 300000, 300000);
```

L'enregistrement est créé sans problème.

3.9 Limitations de modification d'une structure de table

3.9.1 Introduction

Lors de la conception d'une base de données, un développeur teste les structures de la base de données pour déterminer leur exactitude et leur adéquation. Cette tâche peut rarement être effectuée sans charger les tables avec un échantillon. Lorsqu'une table nécessite un changement, par exemple, une colonne additionnelle, il est plus facile de la modifier que de la supprimer pour la recréer avec une nouvelle définition. Les changements qui doivent intervenir ne peuvent parfois pas être réalisés sans procéder aux mêmes modifications sur l'échantillon de données servant aux tests.

3.9.2 Modification d'une clé primaire

Il est souvent difficile de modifier la clé primaire d'une table sans affecter le reste de la base. La raison en est simple. Les tables sont habituellement liées les unes aux autres par l'intermédiaire de déclarations de clés étrangères qui dépendent elles-mêmes de l'existence de clés primaires situées dans d'autres tables. Par conséquent, si vous modifiez la clé primaire, le changement peut se répercuter à travers toute la base. En fait, le système empêche que cela se produise.

Modifier une clé primaire est un processus à deux étapes : supprimer la clé puis la recréer. Le listing suivant illustre ce propos. Imaginez que vous ayez créé à l'origine la table Course avec une clé primaire se composant d'une seule colonne, Course_ID. Après réflexion, vous réalisez que les départements souhaiteraient proposer des cours avec des numéros semblables. Vous devez donc modifier la clé primaire pour qu'elle incorpore à la fois les attributs Course_ID et Department_ID.

```
ALTER TABLE Course ADD CONSTRAINT PK_Course PRIMARY KEY(Course_ID);
```

La table est modifiée.

```
ALTER TABLE Course DROP CONSTRAINT PK_Course
```

La table est modifiée.

```
ALTER TABLE Course ADD CONSTRAINT PK_Course PRIMARY KEY(Course_ID,Department_ID);
```

La table est modifiée.

3.10 Script de création de l'exemple de base de données

Maintenant que vous avez découvert les techniques de base employées pour créer des tables, vous êtes prêt à analyser le script de création de notre exemple de base de données.

La première portion du script est utilisée pour supprimer toutes les tables qui auraient déjà été créées. Si vous tentez de supprimer une table à laquelle se réfère la clé étrangère d'une autre table, le système répondra par un message d'erreur.
Il faut alors respecter un certain ordre pour veiller à ce qu'au moment de la suppression d'une table, celle-ci ne serve pas de référence à une clé étrangère d'une autre table non encore supprimée.
NB : Avec certains systèmes de gestion de bases de données (par exemple Oracle), il est possible d'utiliser la clause CASCADE dans l'instruction DROP TABLE.

```
DROP TABLE Student_Schedule;
DROP TABLE Class;
DROP TABLE Class_Location;
DROP TABLE Schedule_Type_Details;
DROP TABLE Schedule_Type;
DROP TABLE Instructor;
DROP TABLE Course;
DROP TABLE Student;
DROP TABLE Department;


CREATE TABLE Department (
    Department_ID     varchar(20),
    Department_Name    varchar(25) CONSTRAINT NN_Department_Name NOT NULL,
    CONSTRAINT PK_Department PRIMARY KEY(Department_ID)
);


CREATE TABLE Student (
    Student_ID     varchar(20),
    Last_Name       varchar(25) CONSTRAINT NN_Student_Last_Name NOT NULL,
    First_Name      varchar(25) CONSTRAINT NN_Student_First_Name NOT NULL,
    MI              varchar(1),
    Year            varchar(25),
    Street_Address  varchar(25),
    City            varchar(25),
    State           varchar(2),
    Zipcode         varchar(9),
    Telephone       varchar(10),
    Fax             varchar(10),
    Email           varchar(100),
    CONSTRAINT PK_Student PRIMARY KEY(Student_ID)
);


CREATE TABLE Course (
    Course_ID       varchar(5),
    Department_ID    varchar(20) CONSTRAINT NN_Course_Department_ID NOT NULL,
    Title            varchar(60) CONSTRAINT NN_Course_Title NOT NULL,
    Description      varchar(2000),
    Units            tinyint,
    Additional_Fees  smallmoney,
    CONSTRAINT PK_Course
    PRIMARY KEY (Course_ID, Department_ID),
    CONSTRAINT FK_Course_Department_ID
    FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID)
);


CREATE TABLE Instructor (
    Instructor_ID     varchar(20),
    Department_ID      varchar(20)
    CONSTRAINT NN_Instructor_Dept_ID NOT NULL,
    Last_Name          varchar(25) CONSTRAINT NN_Instructor_Last_Name NOT NULL,
    First_Name         varchar(25),
    MI                 varchar(1),
    Position           varchar(25) constraint CK_Instructor_Position
    CHECK (Position in ('ASSISTANT PROFESSOR', 'ASSOCIATE PROFESSOR',
                        'PROFESSOR')),
    Telephone          varchar(10),
    Fax                varchar(10),
    Email              varchar(100),
    CONSTRAINT PK_Instructor PRIMARY KEY (Instructor_ID),
```

```
    CONSTRAINT FK_Instructor_Department_ID
    FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID)
);


CREATE TABLE Schedule_Type (
    Schedule_ID          varchar(20),
    Schedule_Description  varchar(2000),
    CONSTRAINT PK_Schedule_Type PRIMARY KEY (Schedule_ID)
);

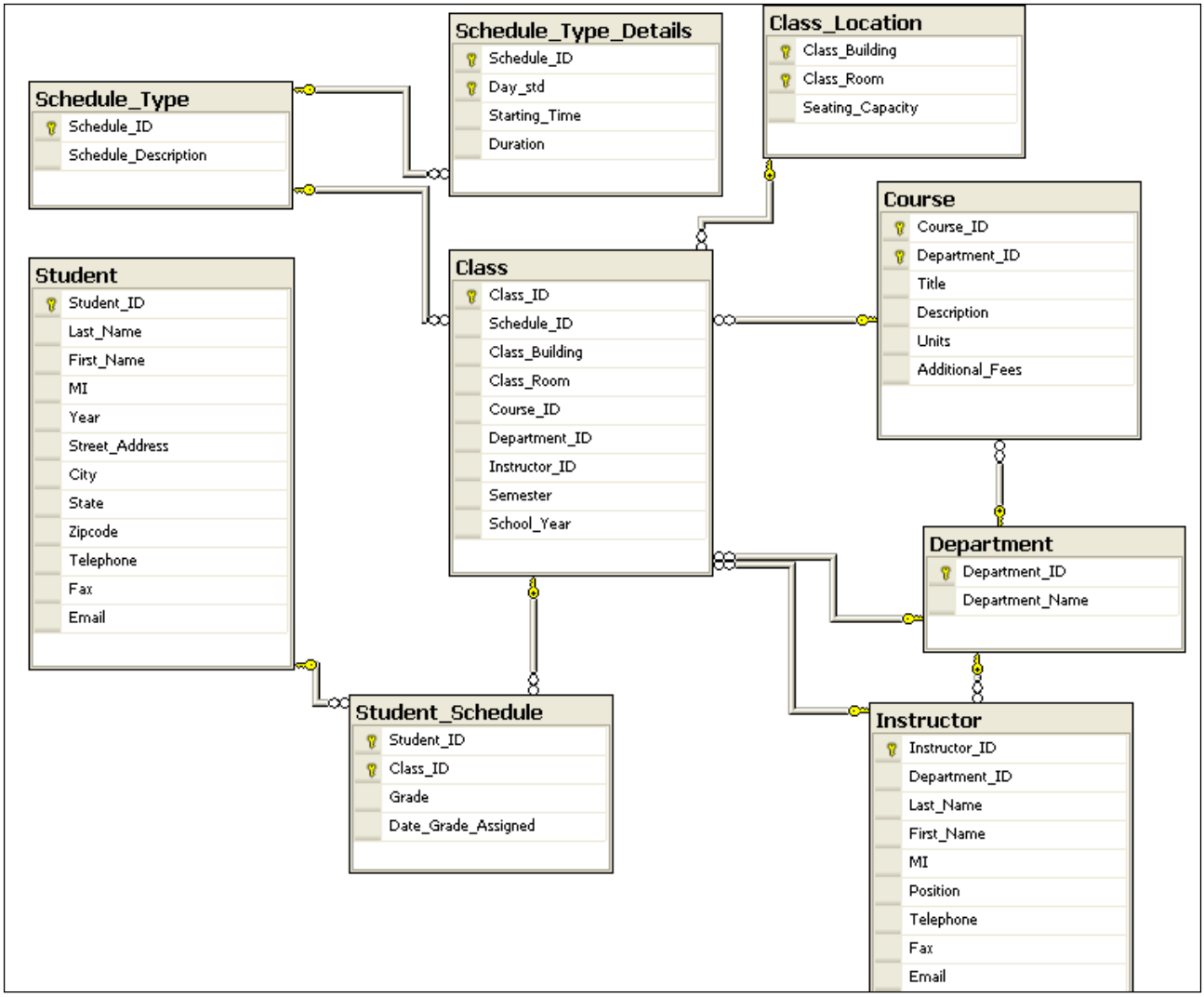

CREATE TABLE Schedule_Type_Details(
    Schedule_ID          varchar(20),
    Day_std                   int
    CONSTRAINT CK_Schedule_Det_Day check (Day_std between 1 and 7),
    Starting_Time         smalldatetime,
    Duration              tinyint,
    CONSTRAINT PK_Schedule_Type_Details PRIMARY KEY (Schedule_ID, Day_std),
    CONSTRAINT FK_Schedule_Det_ID
    FOREIGN KEY (Schedule_ID) REFERENCES Schedule_Type(Schedule_ID)
);


CREATE TABLE Class_Location (
    Class_Building     varchar(25),
    Class_Room          varchar(25),
    Seating_Capacity    int,
    CONSTRAINT PK_Class_Location PRIMARY KEY (Class_Building, Class_Room)
);


CREATE TABLE Class (
    Class_ID           varchar(20),
    Schedule_ID         varchar(20),
    Class_Building      varchar(25),
    Class_Room          varchar(25),
    Course_ID           varchar(5),
    Department_ID       varchar(20),
    Instructor_ID       varchar(20),
    Semester            varchar(6),
    School_Year         smalldatetime,
    CONSTRAINT PK_Class PRIMARY KEY (Class_ID),
    CONSTRAINT FK_Class_Location
    FOREIGN KEY (Class_Building, Class_Room) references
    Class_Location (Class_Building, Class_Room),
    CONSTRAINT FK_Class_Schedule_ID
    FOREIGN KEY (Schedule_ID) references Schedule_Type(Schedule_ID),
    CONSTRAINT FK_Class_Course_ID FOREIGN KEY (Course_ID, Department_ID)
    REFERENCES Course (Course_ID, Department_ID),
    CONSTRAINT FK_Class_Dept_ID FOREIGN KEY (Department_ID) REFERENCES
    Department(Department_ID),
    CONSTRAINT FK_Class_Instructor_ID FOREIGN KEY (Instructor_ID) REFERENCES
    Instructor(Instructor_ID)
);


CREATE TABLE Student_Schedule (
    Student_ID          varchar(20),
    Class_ID             varchar(20),
    Grade                varchar(2)
    CONSTRAINT CK_Grade  CHECK (Grade in ('A', 'A+', 'A-', 'B', 'B+', 'B-',
                               'C', 'C+', 'C-', 'D', 'D+', 'D-',
                               'F', 'F+', 'F-')),
```

```
    Date_Grade_Assigned  smalldatetime,
    CONSTRAINT PK_Student_Schedule PRIMARY KEY (Student_ID, Class_ID),
    CONSTRAINT FK_Student_Schedule_Student_ID FOREIGN KEY (Student_ID) REFERENCES
    Student (Student_ID),
    CONSTRAINT  FK_Student_Schedule_Class  FOREIGN  KEY  (Class_ID)  REFERENCES
    Class(Class_ID)
);
```



3.11 Résumé

Gardez à l'esprit les suggestions suivantes :

- Utilisez l'instruction CREATE TABLE pour créer la structure d'une table.
- Utilisez l'instruction ALTER TABLE pour modifier la structure d'une table existante.
- Définissez les valeurs par défaut pour les colonnes lorsque cela est approprié.
- Chaque table devrait avoir une clé primaire.
- Utilisez des clés étrangères pour assurer l'intégrité référentielle.
- Vous pouvez définir une clé primaire lors de l'utilisation des instructions CREATE TABLE ou ALTER TABLE.
- Vous pouvez supprimer une clé primaire avec l'instruction ALTER TABLE.

- Vous pouvez désactiver ou activer une contrainte avec l'instruction ALTER TABLE.
- Vous pouvez supprimer une contrainte avec l'instruction ALTER TABLE.
- Un index de table peut être défini pour un maximum de 16 colonnes (cela peut varier d'un système de gestion de bases de données à l'autre).
- Un index de table peut être créé pour des valeurs uniques ou non uniques.
- Utilisez une contrainte CHECK lorsque les valeurs acceptées dans une colonne doivent faire partie d'une plage de valeurs déterminée.

4 Modifications des données avec SQL

4.1 Langage de manipulation des données

Les formats de fichiers et les codes spéciaux internes du système d'exploitation sont les premiers éléments que certains programmeurs souhaitent apprendre à propos du système utilisé. Cette approche n'est pas conseillée. Pour vous en convaincre, sachez que :

- Le format des fichiers de bases de données de SQL Server par exemple est la propriété de Microsoft.
- Chaque nouvelle version apporte des changements à ces formats et à ces codes.
- Votre application est plus facile à maintenir si vous vous appuyez sur un standard industriel tel que SQL, plutôt que sur une connaissance quelconque des formats de fichiers. De cette manière, vous assurez une certaine compatibilité de votre application SQL Server avec un autre SGBD.

Pour développer une application de base de données, il est essentiel que vous développiez votre connaissance du langage de requête structuré SQL. C'est un langage puissant qui diffère en de nombreux points des langages traditionnels tels que C et Pascal :

- SQL est un langage non procédural. Vous l'utilisez pour indiquer au système quelles données rechercher ou modifier sans lui indiquer comment réaliser ce travail.
- SQL ne dispose pas d'instructions pour contrôler le flux d'exécution du programme, pour définir une fonction ou exécuter une boucle, ni d'expressions conditionnelles du type **if ... then** ... **else.** Toutefois, le système SQL Server par exemple, fournit un langage procédural appelé Transact SQL qui constitue une extension du langage SQL.
- SQL dispose d'un ensemble assez complet de types de données.

Une similitude entre SQL et un langage de programmation traditionnel est que tous deux proposent généralement plus d'une méthode pour réaliser un même objectif, particulièrement lorsque vous recherchez des informations. Différentes instructions SQL permettant d'obtenir un même résultat peuvent néanmoins différer en efficacité ou en clarté.

4.2 Recherche et modification des données

Pour rappel, les instructions SQL peuvent être réparties en trois catégories :

- Le langage de définition de données, ou LDD, qui définit la structure des données.
- Le langage de manipulation de données, ou LMD, qui recherche et modifie les données.
- Le langage de contrôle de données, ou LCD, qui définit les privilèges d'accès accordés aux utilisateurs de la base de données.

La catégorie LMD comprend quatre instructions de base :

- **INSERT,** qui ajoute des données à une table.
- **UPDATE,** qui modifie les lignes existantes d'une table.
- **DELETE,** qui supprime les lignes d'une table.
- **SELECT,** qui recherche les données d'une table.

Ces instructions sont utilisées le plus souvent par les développeurs d'applications. Les instructions des catégories LDD et LCD sont généralement employées par les concepteurs et les administrateurs pour mettre en place les structures de bases de données utilisées par une application.

4.3 Grammaire SQL

Voici quelques exigences de syntaxe à garder à l'esprit lorsque vous travaillez avec SQL :

- Chaque instruction SQL se termine par un point-virgule.
- Une instruction SQL peut être saisie sur une ligne ou, par souci de clarté, répartie sur plusieurs. La plupart des exemples de ce cours comprennent des instructions fractionnées en portions lisibles.
- SQL ne tient pas compte de la casse ; vous pouvez combiner les majuscules et les minuscules lorsque vous vous référez aux mots clés SQL (tels que **SELECT** et **INSERT),** aux noms de tables et aux noms de colonnes. Toutefois, suivant le système de gestion de bases de données et selon la configuration de ce dernier, l'utilisation des majuscules ou minuscules peut avoir son

importance lorsque vous vous référez au contenu d'une colonne. Dans certains cas, si vous demandez tous les clients dont le nom commence par « a » et que tous les noms de clients sont stockés en majuscules, aucun ne sera fourni en résultat.

NB : Dans le cadre du cours, avec SQL Server, il n'y aura **pas** de différenciation sur la casse.

4.4 Syntaxe de l'instruction INSERT

L'instruction INSERT ajoute des lignes à une table. Avec cette instruction, vous fournissez des valeurs et des expressions littérales à enregistrer sous forme de lignes dans une table.
Le terme INSERT induit les nouveaux utilisateurs de SQL en erreur. Ils s'imaginent qu'ils peuvent déterminer où, dans une table, une ligne est insérée. Souvenez-vous qu'un des points forts des bases de données relationnelles est l'indépendance logique des données qu'elles manipulent, en d'autres termes, une table ne possède aucun ordre implicite. Une nouvelle ligne insérée est placée à un endroit arbitraire dans la table.

Syntaxe de l'instruction INSERT :

L'instruction INSERT peut prendre deux formes. La première est :

```
INSERT INTO nom_table
[(nom_colonne [,nom_colonne] ... [,nom_colonne])]
VALUES
(valeur_colonne[,valeur_colonne] ... [,valeur_colonne])
```

Les variables sont définies ainsi :

- *nom_table* est la table dans laquelle la ligne est insérée.
- *nom_colonne* est une colonne appartenant à *nom_table*.
- *valeur_colonne* est une valeur ou une expression littérale dont le type correspond à celui de *nom_colonne*.

Par exemple, supposez que vous souhaitiez ajouter un nouveau cours à la table Course. Le listing suivant montre l'instruction permettant de le faire.

```
INSERT INTO Course(Course_ID, Department_ID, Title, Description, Units,
Additional_Fees) VALUES('501', 'PSYCH', 'PSYCH IN FILM','Seminar on the portrayal
of psychologists and psychiatrists in film', 6, 3);
```

Notez que le nombre de valeurs constantes ou expressions apparaissant entre parenthèses après le mot clé VALUES doit correspondre au nombre de colonnes spécifiées dans la liste des colonnes.

Le listing suivant contient un exemple de ce qui se produit si six colonnes sont spécifiées, mais seulement cinq valeurs sont fournies. Si vous fournissez un nombre de valeurs différent du nombre de colonnes, le système répond par un message d'erreur. Le système ne dispose évidemment d'aucun moyen pour déterminer quelle est la valeur de colonne manquante.

```
INSERT INTO Course(Course_ID, Department_ID, Title, Description, Units,
Additional_Fees) VALUES('501', 'PSYCH', 'PSYCH IN FILM', 'Seminar on the portrayal
of psychologists and psychiatrists in film', 6);

Msg 109, Niveau 15, État 1, Ligne 1

Il y a plus de colonnes dans l'instruction INSERT que de valeurs spécifiées
dans la clause VALUES. Le nombre de valeurs de la clause VALUES doit être
le même que le nombre de colonnes de l'instruction INSERT.
```

Le listing suivant illustre la situation inverse, moins de colonnes spécifiées que de valeurs données.

```
INSERT INTO Course(Course_ID, Department_ID, Title, Description, Units)
VALUES('501', 'PSYCH', 'PSYCH IN FILM', 'Seminar on the portrayal of psychologists
and psychiatrists in film', 6, 3);
```

```
Msg 110, Niveau 15, État 1, Ligne 1

Il y a moins de colonnes dans l'instruction INSERT que de valeurs spéci-
fiées dans la clause VALUES.
Le nombre de valeurs de la clause VALUES doit être le même que le nombre
de colonnes de l'instruction INSERT.
```

De même si un nom de colonne référencé dans une instruction INSERT est mal orthographié, SQL Server retourne un message d'erreur. Dans l'exemple suivant, la colonne Units est orthographié Unit.

```
INSERT INTO Course(Course_ID, Department_ID, Title, Description, Unit,
Additional_Fees) VALUES('501', 'PSYCH', 'PSYCH IN FILM', 'Seminar on the
portrayal of psychologists and psychiatrists in film',6,3);

Msg 207, Niveau 16, État 1, Ligne 1
Nom de colonne non valide : 'Unit'.
```

4.5 Ajout de lignes avec INSERT

4.5.1 Introduction

Chaque valeur de colonne fournie dans une instruction INSERT doit correspondre à une des formes suivantes :

- un NULL;
- une valeur de constante telle que 3.14159 ou 'BEL';
- une expression contenant des opérateurs et des fonctions, telle que SUBSTRING(Last_Name,1,4).

Dans une instruction INSERT, vous pouvez combiner des valeurs de constantes avec des expressions. Le listing suivant donne un exemple d'utilisation de l'instruction dans lequel les frais additionnels d'un cours sont évalués à 10 % du résultat donné par la multiplication du nombre total d'unités par 250 $.

```
INSERT INTO Course(Course_ID, Department_ID, Title, Description, Units, Addi-
tional_Fees) VALUES('501', 'PSYCH', 'PSYCH IN FILM', 'Seminar on the portrayal of
psychologists and psychiatrists in film', 6, 0.10 * 3 * 250);

SELECT Additional_Fees
FROM Course
WHERE Course_ID = '501' and Department_ID = 'PSYCH';

ADDITIONAL_FEES
---------------
      75
```

4.5.2 Cohérence entre le type de la valeur insérée et celui de la colonne

À quelques exceptions près, qui seront présentées plus loin, les types de données d'une colonne et de sa valeur doivent être identiques. Insérer une chaîne alphanumérique dans une colonne numérique n'a pas de sens.

Exemple : incohérence de type de données entre une colonne et sa valeur dans l'instruction INSERT.

```
INSERT INTO Course(Course_ID, Department_ID, Title, Description, Units,
Additional_Fees) VALUES ('501', 'PSYCH', 'PSYCH IN FILM', 'Seminar on the
portrayal of psychologists and psychiatrists in film',6, 'Seventy-five
dollars');

Msg 293, Niveau 16, État 0, Ligne 1
Impossible de convertir une valeur char en smallmoney. La syntaxe de la
valeur char est incorrecte.
```

Dans l'exemple précédent, le système a retourné un code d'erreur, car la chaîne « Seventy-five dollars » ne peut pas être stockée dans une colonne définie pour recevoir un nombre (ou une donnée monétaire). Toutefois, si la chaîne représente un nombre valide, l'instruction peut être traitée avec succès (voir ci-après). Cela constitue cependant une exception à la règle stipulant qu'une colonne et la valeur qui lui est assignée doivent être de type identique.

Exemple : conversion d'une chaîne en valeur numérique dans une instruction INSERT.

```
INSERT INTO Course(Course_ID, Department_ID, Title, Description, Units,
Additional_Fees) VALUES('501', 'PSYCH', 'PSYCH IN FILM','Seminar on the
portrayal of psychologists and psychiatrists in film', 3, '75');
```

4.5.3 Utilisation de pseudo-colonnes et de fonctions en tant que valeurs

Le système fournit un ensemble de fonctions appelées *pseudo-colonnes*. Elles sont ainsi nommées, car elles se comportent de prime abord comme des colonnes. Les deux pseudo-colonnes couramment utilisées sont :

- USER, qui retourne le nom utilisateur actuellement connecté à la base de données.
- GETDATE, qui retourne la date et l'heure courantes.

Vous pouvez utiliser les pseudo-colonnes dans une instruction INSERT pour attribuer une valeur à une colonne. Par exemple, USER et GETDATE peuvent servir à enregistrer le nom de l'utilisateur qui a créé la ligne, ainsi que la date et l'heure de l'insertion (cf. ci-après).

```
INSERT INTO Student_Schedule(Student_ID, Class_ID, Grade, Date_Grade_Assigned,
Created_Username, Created_Timestamp) VALUES ('10231311', '104200', 'B',
'02/07/2009', USER, GETDATE());
```

NB : Ne pas oublier les parenthèses ( ) derrière GETDATE qui est une fonction.

Vous ne pouvez assigner la pseudo-colonne USER qu'à une colonne de type chaîne. Vous pouvez manipuler ces pseudo-colonnes (USER, GETDATE(), ...) à l'aide de fonctions et d'opérateurs. Dans le listing qui suit, une constante chaîne est concaténée avec le nom de l'utilisateur courant avant d'être stockée dans la colonne Created_Username.

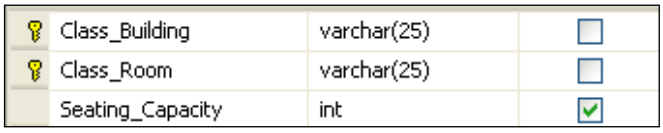
```
INSERT INTO Student_Schedule(Student_ID, Class_ID, Grade, Date_Grade_Assigned,
Created_Username, Created_Timestamp) VALUES('10231311', '104500', 'B-',
'03/07/2009','User is: ' + USER, GETDATE());

SELECT Student_ID, Grade, Created_Username, Created_Timestamp
FROM Student_Schedule
WHERE Student_ID = '10231311' and Class_ID = '104500';
```

L'instruction INSERT spécifie deux pseudo-colonnes : USER et GETDATE. L'instruction SELECT montre qu'elles ont bien retourné les valeurs attendues dans l'instruction INSERT.

4.5.4 Spécification des colonnes dans une instruction INSERT

Si vous examinez attentivement la description de la syntaxe de l'instruction INSERT dans le manuel de référence en ligne du langage SQL, vous noterez que la liste des colonnes est un élément optionnel. Par conséquent, si vous ne spécifiez pas le nom des colonnes devant recevoir les valeurs, le système utilise par défaut toutes les colonnes. De plus, l'ordre des colonnes que le système utilise est l'ordre dans lequel elles ont été spécifiées lors de la création de la table. Il s'agit du même ordre que celui dans lequel les colonnes sont présentées lorsque l'on se trouve en mode création notamment.




```
INSERT INTO Class_Location VALUES('250', 'MACLAREN COMPLEX', 75);
```

```
SELECT Class_Building, Class_Room, Seating_Capacity
FROM Class_Location
WHERE Class_Building = '250';
```

	Class_Building	Class_Room	Seating_Capacity
1	250	MACLAREN COMPLEX	75

L'utilisation de cette syntaxe peut être source de problèmes :

- La définition d'une table peut changer. Le nombre de colonnes pouvant diminuer ou augmenter, l'insertion peut résulter en un échec.
- L'instruction INSERT peut être difficile à comprendre. Les valeurs peuvent être mal différenciées les unes des autres et être inversées accidentellement. Tenter de résoudre un problème d'insertion sans être en mesure de visuellement faire correspondre une colonne avec sa valeur est risqué.
- L'instruction INSERT peut réussir, mais des données erronées peuvent avoir été introduites dans la table. Cela peut se produire lorsque deux valeurs contiguës de même type sont inversées par inadvertance. Ce cas peut engendrer des problèmes sérieux, car le système ne retournerait pas de message d'erreur.

4.6 Utilisation d'une sous-requête : syntaxe de l'instruction INSERT incluant une sous-requête

Jusqu'à présent, les exemples illustraient l'utilisation d'une instruction INSERT pour ajouter une simple ligne à une table. L'exemple suivant montre comment exécuter une insertion sans spécifier directement de valeurs. Les valeurs seront extraites via une sous-requête.

Une autre forme de l'instruction INSERT remplace la liste des valeurs de colonnes par une instruction SELECT introduite ainsi :

```
INSERT INTO nom_table
[(nom_colonne [,nom_colonne] ... [,nom_colonne])]
instruction_SELECT;
```

Les variables sont définies ainsi

- *nom_table* est la table dans laquelle la ligne est insérée.
- *nom_colonne* est une colonne appartenant à *nom_table*.
- *instruction_SELECT* est une sous-requête SELECT valide.

Supposez qu'une table soit utilisée pour stocker des informations à propos des enseignants ne travaillant plus à l'université Flugle. Pour célébrer le centenaire de l'université, les anciens enseignants sont remis en activité.

Note :
La table Inactive_Instructor n'existe pas dans notre exemple de base de données. Néanmoins, pour ajouter les enseignants retraités à la table Instructor, il faudrait utiliser une sous-requête pour sélectionner les lignes de la table Inactive_Instructor et les insérer dans la table Instructor (voir listing ci-dessous).

Exemple : insertion de lignes avec une sous-requête.

```
INSERT INTO Instructor(Instructor_ID, Last_Name, First_Name, Department_ID)
SELECT Instructor_ID, Last_Name, First_Name, Department_ID
FROM Inactive_Instructor;
```

Pour utiliser une sous-requête avec une instruction INSERT, le nombre de colonnes auxquelles se réfère l'instruction d'insertion doit être égal au nombre de colonnes spécifiées dans la liste de sélection de la sous-requête SELECT.

4.7 Modification de données avec UPDATE

Si vous souhaitez modifier des données existantes dans une base de données, vous pouvez utiliser l'instruction UPDATE. Cette instruction permet de mettre à jour une ou plusieurs lignes d'une table.

4.7.1 Syntaxe de l'instruction UPDATE

Comme pour l'instruction INSERT, la syntaxe de l'instruction UPDATE est beaucoup plus simple que celle de l'instruction SELECT :

```
UPDATE nom_table
SET nom_colonne = expression   [, nom_colonne = expression] ...
     [, nom_colonne = expression]
[WHERE condition];
```

Les variables sont définies ainsi :

- *nom_table* est la table à modifier.
- *nom_colonne* est une colonne dans la table à modifier.
- *expression* est une expression SQL valide.
- *condition* est une condition SQLvalide.

Comme vous pouvez le voir, l'instruction UPDATE se réfère à une seule table et affecte une expression à au moins une colonne. La clause WHERE est optionnelle, et si elle n'est pas spécifiée dans l'instruction, l'affectation des valeurs portera sur toutes les lignes de la table.

4.7.2 Modification de valeurs dans plusieurs colonnes

Comme vous avez pu le constater dans la présentation de la syntaxe de l'instruction, une requête UPDATE peut modifier les valeurs de plusieurs colonnes dans une table. Le listing qui suit illustre la modification de valeurs dans deux colonnes : Position et Telephone.

```
UPDATE Instructor
SET Position = 'PROFESSOR',
    Telephone = '8055551212';
```



ATTENTION : Avec la requête précédente, il faut constater que tous les enregistrements sont modifiés étant donné l'absence de condition (WHERE).

4.8 Permutation de valeurs avec UPDATE

Le listing suivant présente une requête qui sélectionne les valeurs des attributs Instructor_ID, Telephone, et Fax de la table Instructor. Vous pouvez intervertir les numéros de téléphone et de fax de la table au moyen d'une simple instruction UPDATE. Il n'est pas nécessaire de stocker au préalable les numéros dans des variables temporaires, comme vous le feriez avec un langage de programmation. La seconde requête de la table Instructor montre que les valeurs des deux colonnes ont été permutées avec succès.

```
SELECT Instructor_ID, Telephone, Fax
FROM Instructor
ORDER BY Instructor_ID;

INSTRUCTOR_ID        TELEPHONE  FAX
-------------------- ---------- ----------
A612
B331
```

```
B391
D201                 8055550131 8055559444
D944
E301
E405                 8055554455
E491
G331
J505
L391
M101                 8055550123 8055550321
R983
S131
T149
W490
Y561                 8055550123

UPDATE Instructor
SET Telephone = Fax,
    Fax = Telephone;

(17 ligne(s) affectée(s))

SELECT Instructor_ID, Telephone, Fax
FROM Instructor
ORDER BY Instructor_ID;

INSTRUCTOR_ID        TELEPHONE  FAX
-------------------- ---------- ----------
A612
B331
B391
D201                 8055559444 8055550131
D944
E301
E405                            8055554455
E491
G331
J505
L391
M101                 8055550321 8055550123
R983
S131
T149
W490
Y561                            8055550123
```

4.9 Utilisation d'une sous-requête avec UPDATE

Jusqu'ici, ce chapitre a présenté l'utilisation des expressions conditionnelles dans la clause WHERE pour restreindre le nombre de lignes affectées par une instruction UPDATE. Voyons comment employer une sous-requête avec cette instruction.
Supposez que vous souhaitiez réduire de 10 dollars les frais additionnels d'un cours dans le cas où ils excèdent la moyenne calculée. Cela pourrait être réalisé avec une simple instruction UPDATE (cf. ci-après). Notez que la sous-requête doit être placée entre parenthèses.

```
SELECT Department_ID, Course_ID, Additional_Fees
FROM Course
ORDER BY Department_ID, Course_ID;
```

```
DEPART COURS ADDITIONAL_FEES
------ ----- ---------------
ANTHRO 101                 0
ANTHRO 174                55
ANTHRO 189               7.5
BIO    101                55
BIO    177                65
BIO    178                70
ECON   101                25
ECON   189               750
ECON   199                 0
ENG    101                75
ENG    102                45
ENG    103                35
ENG    199                45
ENGL   101                 0
ENGL   189                 0
ENGL   192                 0
ENGL   193                 0
HIST   115                 0
HIST   184                 0
HIST   199                 0
MATH   101                 0
MATH   189                 0
MATH   50                  0
MATH   51                 10
PHILO  101                 0
PHILO  174                 0
PHILO  198                 0
PSYCH  101                25
PSYCH  181                75
PSYCH  183                45
PSYCH  185                20
PSYCH  501                75
```

```
UPDATE Course
SET Additional_Fees = Additional_Fees - 10
WHERE Additional_Fees > (SELECT avg(Additional_Fees) FROM Course);
```

```
SELECT Department_ID, Course_ID, Additional_Fees
FROM Course
ORDER BY Department_ID, Course_ID;
```

```
DEPART COURS ADDITIONAL_FEES
------ ----- ---------------
ANTHRO 101                 0
ANTHRO 174                45
ANTHRO 189               7.5
BIO    101                45
BIO    177                55
BIO    178                60
ECON   101                25
ECON   189               740
ECON   199                 0
ENG    101                65
ENG    102                45
ENG    103                35
ENG    199                45
ENGL   101                 0
ENGL   189                 0
```

```
ENGL   192                0
ENGL   193                0
HIST   115                0
HIST   184                0
HIST   199                0
MATH   101                0
MATH   189                0
MATH   50                 0
MATH   51                10
PHILO  101                0
PHILO  174                0
PHILO  198                0
PSYCH  101               25
PSYCH  181               65
PSYCH  183               45
PSYCH  185               20
PSYCH  501               65
```

4.10 Suppression de données avec DELETE

L'instruction DELETE supprime les lignes d'une table. Il n'est pas nécessaire de connaître l'ordre physique des lignes dans une table pour utiliser cette instruction. Le système utilise le critère de la clause WHERE pour déterminer les lignes à supprimer, et le moteur de la base de données se charge de déterminer l'emplacement interne des données.

4.10.1 Syntaxe de l'instruction DELETE

Cette instruction possède la syntaxe la plus simple des quatre instructions du LMD.

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

Les variables sont définies ainsi :

- *nom_table* est la table contenant les valeurs à supprimer.
- *condition* est une condition SQL valide.

Vous pouvez observer que la syntaxe du langage SQL n'est pas cohérente. Par exemple, la syntaxe de l'instruction UPDATE (UPDATE nom_table) diffère de la syntaxe de l'instruction DELETE (DELETE FROM nom_table). SQL possède beaucoup d'autres particularités qui ne risquent pas de disparaître à court terme. Pour tirer parti de la puissance de SQL, concentrez-vous sur l'apprentissage de sa syntaxe et sur sa mise en pratique au moyen de nombreux exercices. Le listing qui suit montre un exemple simple d'instruction DELETE.

```
DELETE FROM Class
WHERE Department_ID = 'BIO';

(1 ligne(s) affectée(s))
```

4.10.2 Suppression de toutes les lignes d'une table avec TRUNCATE TABLE

Lors de la conception d'une application, il est possible que vous ayez à supprimer toutes les lignes d'une table. Si elle contient un grand nombre de lignes, l'instruction DELETE peut se révéler inefficace. Vous pouvez alors recourir à l'instruction TRUNCATE TABLE qui réalise cette tâche plus rapidement.

En résumé, l'instruction TRUNCATE TABLE supprime toutes les lignes d'une table sans enregistrer les suppressions de lignes individuelles dans le journal. L'instruction TRUNCATE TABLE est semblable à l'instruction DELETE sans la clause WHERE. Cependant, l'instruction TRUNCATE TABLE est plus rapide et utilise moins de ressources système et de ressources du journal des transactions.

L'instruction TRUNCATE TABLE n'est pas une instruction du LMD. Par conséquent, si vous l'utilisez vous ne pourrez pas changer d'avis et effectuer un ROLLBACK pour récupérer les lignes perdues. Cette instruction vous propose un aller sans retour.

La syntaxe de l'instruction est la suivante :

TRUNCATE TABLE *nom_table*

La variable est définie ainsi :

- *nom_table* est la table sur laquelle porte la suppression générale.

4.11 Accès simultanés au système d'information

Un système d'information idéal permet la connexion concurrente de plusieurs utilisateurs. Des mécanismes doivent cependant veiller à ce que les modifications des utilisateurs n'entrent pas en conflit. Si un administrateur du département d'anthropologie change la description d'un cours, un autre administrateur du même département ne doit pas être en mesure de modifier la même information tant que le premier administrateur n'a pas fini de valider ses changements.

Par ailleurs, le fait qu'une personne modifie des informations relatives à un cours ne devrait pas empêcher qui que ce soit de les consulter, ou d'apporter des changements à un autre cours.

4.12 Attribution d'une valeur NULL

Voici les diverses raisons pour lesquelles une colonne peut recevoir la valeur NULL :

- La colonne ne reçoit pas de valeur lors d'une insertion et ne possède pas de valeur par défaut.
- La colonne reçoit de façon explicite la valeur NULL lors de l'insertion.
- La valeur de colonne est modifiée pour NULL lors d'une opération de modification.

4.12.1 Valeur NULL avec INSERT

Vous pouvez attribuer explicitement une valeur NULL à une colonne dans une instruction INSERT.

```
INSERT INTO Instructor(Instructor_ID, Department_ID, Last_Name, Telephone)
VALUES('P331', 'ANTHRO', 'POULSON', NULL);

(1 ligne(s) affectée(s))
```

4.12.2 Valeur NULL avec UPDATE

Vous pouvez utiliser l'instruction UPDATE pour modifier une colonne et lui affecter une valeur NULL. Le listing figurant ci-après en donne un exemple. Après avoir attribué la valeur NULL à la colonne Telephone pour l'enseignant identifié par 'Y561 ', vous pouvez vérifier que le système a bien effectué le changement.

```
SELECT Last_Name, Telephone
FROM Instructor
WHERE Instructor_ID = 'Y561';

LAST_NAME                TELEPHONE
------------------------ ----------
YOUNG                    8055550123

UPDATE Instructor
SET Telephone = NULL
WHERE Instructor_ID = 'Y561';

(1 ligne(s) affectée(s))
```

```
SELECT Last_Name, Telephone
FROM Instructor
WHERE Instructor_ID = 'Y561';

LAST_NAME                 TELEPHONE
------------------------- ----------
YOUNG
```

L'incohérence de la syntaxe de SQL donne lieu à certaines critiques. Dans l'instruction UPDATE, une valeur NULL est assignée à une colonne au moyen du signe égal '='. Toutefois, il est recommandé d'utiliser le mot réservé IS (ou IS NOT) à la place du signe égal dans la clause WHERE d'une instruction SELECT, si vous devez tester la présence d'une valeur NULL dans une colonne. Vous pouvez ainsi obtenir le listing suivant.

```
UPDATE Student
SET Fax = NULL
WHERE Fax is not null;

(3 ligne(s) affectée(s))
```

4.12.3 Valeur par défaut et valeur NULL

Lorsque vous créez une table, spécifiez une colonne comme obligatoire en ajoutant NOT NULL après le type de données de la colonne. Une colonne de ce type doit recevoir une valeur chaque fois qu'une ligne est insérée dans la table. Si vous tentez une insertion sans spécifier de valeur pour une colonne obligatoire, le système le signale par un message d'erreur. Par exemple, chaque cours à l'université Flugle doit posséder un identifiant, être associé à un département donné, et avoir un titre. Le listing qui suit contient une instruction INSERT qui tente d'ajouter une ligne à la table Course sans spécifier de valeur pour la colonne Titre qui est définie comme NOT NULL.

```
INSERT INTO Course(Course_ID, Department_ID) VALUES('901', 'ANTHRO');

Msg 515, Niveau 16, État 2, Ligne 1
Impossible d'insérer la valeur NULL dans la colonne 'Title', table
'Base_Essais.dbo.Course'. Cette colonne n'accepte pas les valeurs NULL.
Échec de INSERT.
L'instruction a été arrêtée.
```

4.13 Transaction : unité logique de travail

4.13.1 Présentation

Un autre concept puissant à comprendre est celui de transaction.

Définition :
Une transaction est définie comme une unité logique de travail: un ensemble de changements dans une base de données sur une ou plusieurs tables, prévu pour réaliser une tâche spécifique.

Sous SQL Server, le début d'une transaction dépend du mode de gestion des transactions utilisé.

Transactions explicites

- Une transaction explicite démarre lorsque l'instruction BEGIN TRANSACTION est exécutée.
- Dès ce moment, toutes les opérations suivantes (modifications, suppressions, insertions) sont considérées comme faisant partie de cette transaction, jusqu'à ce qu'une commande COMMIT (validation) ou ROLLBACK (annulation) soit exécutée.

Transactions automatiques (autocommit)

- Par défaut, SQL Server exécute chaque commande individuelle dans sa propre transaction : c'est le mode « autocommit ».
- Ainsi, chaque requête DML (INSERT, UPDATE, DELETE) déclenche une transaction qui est automatiquement validée après exécution, sauf en cas d'erreur.

Transactions implicites

- Si le mode IMPLICIT_TRANSACTIONS est activé (SET IMPLICIT_TRANSACTIONS ON), une nouvelle transaction débute automatiquement à l'exécution d'une commande DML ou DDL (par exemple, INSERT, UPDATE, DELETE, SELECT, CREATE, etc.).
- Après validation (COMMIT) ou annulation (ROLLBACK), SQL Server attend la prochaine commande appropriée pour démarrer une nouvelle transaction implicite.

Résumé :

- Dans le cas standard (explicite), la transaction commence à l'exécution de BEGIN TRANSACTION.
- Sinon, en mode par défaut (autocommit), chaque opération individuelle fait l'objet de sa propre transaction automatiquement validée.
- En mode implicite, une transaction commence dès la première commande DML/DDL après que le mode a été activé.

4.13.2 Validation d'une transaction avec COMMIT

Vous pouvez vous représenter une transaction comme une modification que vous effectuez dans un document en utilisant votre traitement de texte favori. Vous pouvez effectuer plusieurs changements et choisir de les annuler ou plus simplement de quitter le programme sans les enregistrer. En revanche, lorsque vous donnez l'ordre au programme d'enregistrer le fichier, vos modifications deviennent permanentes dans le fichier sur le disque.

La validation d'une transaction est identique à l'enregistrement d'un fichier dans un traitement de texte. Elle est réalisée au moyen de l'instruction COMMIT TRANSACTION (ou COMMIT TRAN ou COMMIT), et rend permanents tous les changements qui auront été effectués dans la base de données depuis le début de la transaction. Vous ne pouvez valider que les changements que vous avez effectués personnellement, l'instruction COMMIT TRANSACTION d'un utilisateur n'a aucune incidence sur les modifications faites par un autre utilisateur.

4.13.3 Annulation d'une transaction avec ROLLBACK TRANSACTION

L'instruction ROLLBACK TRANSACTION (ou ROLLBACK TRAN ou ROLLBACK) a le même effet qu'une commande d'annulation d'un traitement de texte, avec une différence majeure : l'instruction ROLLBACK TRANSACTION défait toutes les modifications effectuées par l'utilisateur dans une base de données depuis la dernière transaction validée ou depuis le début de la session de base de données.

Analysez l'interaction entre les instructions COMMIT TRANSACTION et ROLLBACK TRANSACTION avec l'exemple du listing suivant. La requête sur la Table_1 retourne quatre lignes. Après suppression des lignes de la table aucune ligne ne peut être retournée avec la recherche suivante. L'instruction ROLLBACK TRANSACTION est alors utilisée, et une dernière interrogation permet de constater que la table a été rétablie dans son état initial.

```
SELECT * FROM table_1;

BEGIN TRANSACTION;

DELETE FROM table_1;

(4 ligne(s) affectée(s))
```

```
SELECT * FROM table_1;

ROLLBACK TRANSACTION;   Autre syntaxe possible : ROLLBACK

Commande(s) réussie(s).

SELECT * FROM table_1;
```

4.13.4 Points d'enregistrement

Pour les transactions impliquant l'exécution de plusieurs instructions SQL, vous pouvez envisager l'utilisation de points d'enregistrement (savepoints) comme étapes intermédiaires de la transaction. Il s'agit d'étiquettes au sein d'une transaction qui englobent un sous-ensemble de modifications apportées par la transaction. Le listing suivant illustre comment déclarer un point d'enregistrement, nommé *null_fax_numbers* dans l'exemple.

```
BEGIN TRANSACTION;

SAVE TRANSACTION null_fax_numbers;

Commande(s) réussie(s).
```

Généralement, on utilise un point d'enregistrement en même temps qu'une instruction ROLLBACK TRANSACTION afin de défaire une transaction jusqu'à une phase intermédiaire déterminée par ce point d'enregistrement.

La syntaxe d'utilisation de l'instruction ROLLBACK est la suivante :

```
ROLLBACK TRANSACTION [point_sauvegarde];
```

La variable est définie ainsi :

- ✓ *point_sauvegarde* est le point d'enregistrement déclaré au préalable.

Analysez l'exemple montré dans le listing ci-après. Imaginez que votre application comporte une transaction modifiant trois tables : Table_1, Table_2 et Table_3. Si le système retourne un code d'erreur au moment d'une modification dans Table_2, vous pouvez défaire la transaction jusqu'au point de sauvegarde qui est ultérieur à la modification dans Table_1.

```
BEGIN TRANSACTION;

UPDATE table_1
set table_1_col = 11;

(4 ligne(s) affectée(s))

SAVE TRANSACTION table_1_UPDATE;

Commande(s) réussie(s).

DELETE FROM table_2;

(3 ligne(s) affectée(s))

ROLLBACK TRANSACTION table_1_UPDATE;

Commande(s) réussie(s).
```

```
SELECT * FROM table_2;
```

De plus, l'annulation de la transaction n'a pas eu d'effets sur les mises à jour réalisées dans la table « table_1 » car l'annulation a été exécutée jusqu'au point de sauvegarde.

NB : Les points d'enregistrement doivent être utilisés avec précaution, car ils apportent un niveau de complexité supplémentaire à une application. Soyez certain de la définition de vos transactions avant de déclarer des points d'enregistrement.

4.14 Résumé

Ce chapitre a décrit l'utilisation d'instructions SQL fondamentales :

- INSERT ajoute des lignes à une table.
- UPDATE modifie des lignes existantes d'une table.
- DELETE supprime des lignes d'une table.

Vous pouvez aussi associer une sous-requête à ces instructions.

L'instruction TRUNCATE TABLE supprime de façon irréversible toutes les lignes d'une table.

Une transaction de base de données est un ensemble de changements apportés à une ou plusieurs tables de la base et qui constitue une unité logique de travail. Vous pouvez utiliser l'instruction COMMIT TRANSACTION pour valider la transaction et rendre les modifications permanentes. Inversement, l'instruction ROLLBACK TRANSACTION permet d'annuler la transaction.

5 Sélection de données en utilisant le langage SQL

5.1 Syntaxe de l'instruction SELECT

5.1.1 Introduction

Des quatre instructions du LMD, **SELECT** est celle qui est exécutée le plus souvent dans une application réelle, car les enregistrements sont lus plus généralement qu'ils ne sont modifiés. Cette instruction peut aussi apparaître sous forme de sous-requête dans une instruction **UPDATE, INSERT,** ou **DELETE,** mais ce sujet sera traité ultérieurement.

L'instruction **SELECT** est un outil puissant et sa syntaxe, simple au départ, peut devenir complexe en raison des nombreuses possibilités qui vous sont offertes pour former une instruction valide en combinant les tables, les colonnes, les fonctions et les opérateurs. Par conséquent, au lieu d'examiner la syntaxe complète de cette instruction, cette section débute avec quelques exemples illustrant son utilisation.

Dans sa forme la plus simple, l'instruction contient les deux éléments suivants :

- La liste de sélection, qui est une liste de colonnes ou d'expressions faisant l'objet de la requête.
- La clause FROM, qui indique la table au sein de laquelle les informations sont recherchées.

5.1.2 Instruction SELECT simple

Le listing suivant présente une instruction SELECT simple. La requête ne recherche que les numéros d'identification des étudiants contenus dans la colonne Student_ID de la table Student.

```
SELECT Student_ID
FROM Student;
```

Si vous souhaitez récupérer à la fois l'identifiant et le nom de l'étudiant, il suffit d'indiquer les colonnes dans l'ordre désiré.

```
SELECT Student_ID, Last_Name
FROM Student;
```

Si vous souhaitez extraire toutes les colonnes d'une table, vous pouvez utiliser le caractère générique SQL, l'astérisque (*).

5.1.3 Liste de sélection

Si la liste de sélection contient plusieurs colonnes, ces dernières doivent être séparées par des virgules. La liste peut aussi contenir des expressions valides contenant ou non des colonnes. De plus, une même colonne peut être utilisée plusieurs fois dans la liste.

```
SELECT Student_ID, Student_ID
FROM Student;
```

5.1.4 Résultats retournés par une instruction SELECT

Les résultats retournés par toute instruction **SELECT** forment une table temporaire. Chaque enregistrement récupéré est une ligne dans cette table temporaire, et chaque élément de la liste de sélection est une colonne. Si une requête ne retourne aucun enregistrement, on peut penser que la table temporaire est vide. Ce comportement est un principe fondamental du modèle relationnel.

5.1.5 Utilisation d'expressions dans une instruction SELECT

Outre la spécification de colonnes, vous pouvez aussi spécifier des expressions dans la liste de sélection. Celles-ci entrent dans les mêmes catégories de types de données que les colonnes : caractère, numérique, et date. En utilisant des opérateurs, des fonctions intégrées et des constantes, vous pouvez créer des expressions complexes pour satisfaire les besoins de votre application.
Gardez à l'esprit que le système considère chaque élément de la liste de sélection comme une colonne séparée, même si une expression référence plusieurs colonnes.

5.1.6 Opérateurs arithmétiques et logiques

Les opérateurs arithmétiques et logiques utilisés dans SQL sont similaires à ceux que le langage C emploie :

Description	Opérateur
Addition	+
Soustraction	-
Multiplication	*
Division	/
Inférieur à	<
Inférieur ou égal à	<=
Supérieur à	>
Supérieur ou égal à	>=
Egal à	=
Différent de	<>

À titre d'exemple, prenons la table Course qui contient les unités de chaque cours. Si le coût actuel par unité est de 250 dollars, le coût du cours est obtenu à l'aide de l'instruction **SELECT** suivante.

```
SELECT Title, Units*250 as COUT_TOTAL
FROM Course;
```

```
TITLE                                    UNITS*250
---------------------------------------- ---------
INTRO TO ANTHROPOLOGY                            750
PRE-CALCULUS                                     750
GENERAL CALCULUS                                 750
NUMBER THEORY                                    750
ADVANCED ARITHMETIC                              750
INTRO TO PSYCHOLOGY                              750
ABNORMAL PSYCHOLOGY                              750
EARLY AMERICAN HISTORY                           750
MODERN EUROPEAN HISTORY                          750
INTRO TO BIOLOGY                                 750
INTRO TO ECONOMICS                               750
MONETARY POLICY                                  750
INTRO TO PHILOSOPHY                              750
INTRO TO ENGLISH LIT                             750
MODERN ENGLISH LIT                               750
INTRO TO STRUCTURES                              750
MODERN PHILOSOPHY                                750
ANCIENT GREEK PHILOSOPHERS                       750
MAMMALIAN ANATOMY                                750
INVERTEBRATE ANATOMY                             750
…
```

5.1.7 Opérateurs pour chaînes

Un des opérateurs pour chaînes les plus importants en SQL est celui de concaténation +. La syntaxe SQL requiert que les constantes de type chaîne apparaissent entre guillemets simples. Cet opérateur

permet de concaténer plusieurs chaînes. Suivant les systèmes de gestion de bases de données, cet opérateur peut différer: &, ||, ...

```
SELECT Last_Name + ': ' + Position
FROM Instructor;
```

```
HITCHCOCK: PROFESSOR
DANIELS: ASSOCIATE PROFESSOR
EDWARDS: PROFESSOR
CHANG: ASSISTANT PROFESSOR
JASON: ASSOCIATE PROFESSOR
TORRES: PROFESSOR
ANGELO: ASSOCIATE PROFESSOR
RICHARDSON: ASSISTANT PROFESSOR
PARKER: PROFESSOR
CHERNOW: ASSOCIATE PROFESSOR
CHU: PROFESSOR
WEISS: PROFESSOR
RESTON: ASSOCIATE PROFESSOR
BILLINGS: ASSISTANT PROFESSOR
YOUNG: ASSOCIATE PROFESSOR
NILAND: ASSOCIATE PROFESSOR
BATES: ASSISTANT PROFESSOR
```

Voici plusieurs raisons pour lesquelles vous pourriez avoir besoin de concaténer des chaînes :

- **Pour incorporer des chaînes dans les valeurs retournées par la requête.** Vous voulez envoyer une lettre type à vos clients en combinant la formule de politesse, un espace blanc, et le nom du client :

  ```
  SELECT Salutation +   '   '   + Last_Name
  FROM Customer
  ORDER BY Last_Name;
  ```

- **Pour combiner des chaînes.** Votre application peut nécessiter l'extraction d'une sous-chaîne contenue dans une colonne pour la combiner avec celle d'une autre colonne avant d'afficher le résultat.
- **Pour créer de nouvelles valeurs qui peuvent être assignées à une colonne.** Dans le processus d'implémentation de votre application, il se peut que vous ayez besoin de convertir des données existantes d'un format à un autre.

Vous pouvez utiliser l'opérateur de concaténation avec plus de deux chaînes (voir listing ci-dessous).

```
SELECT Last_Name + ', ' + Position + ', can be contacted at ' + Email
FROM Instructor;
```

```
HITCHCOCK, PROFESSOR, can be contacted at BHITCHCOCK@FLUGLE.EDU
DANIELS, ASSOCIATE PROFESSOR, can be contacted at LDANIELS@FLUGLE.EDU
EDWARDS, PROFESSOR, can be contacted at SEDWARDS@FLUGLE.EDU
CHANG, ASSISTANT PROFESSOR, can be contacted at RCHANG@FLUGLE.EDU
JASON, ASSOCIATE PROFESSOR, can be contacted at JJASON@FLUGLE.EDU
TORRES, PROFESSOR, can be contacted at PTORRES@FLUGLE.EDU
ANGELO, ASSOCIATE PROFESSOR, can be contacted at AANGELO@FLUGLE.EDU
RICHARDSON, ASSISTANT PROFESSOR, can be contacted at NRICHARDSON@FLUGLE.EDU
PARKER, PROFESSOR, can be contacted at WPARKER@FLUGLE.EDU
CHERNOW, ASSOCIATE PROFESSOR, can be contacted at BCHERNOW@FLUGLE.EDU
CHU, PROFESSOR, can be contacted at SCHU@FLUGLE.EDU
WEISS, PROFESSOR, can be contacted at RWEISS@FLUGLE.EDU
RESTON, ASSOCIATE PROFESSOR, can be contacted at RRESTON@FLUGLE.EDU
BILLINGS, ASSISTANT PROFESSOR, can be contacted at BBILLINGS@FLUGLE.EDU
YOUNG, ASSOCIATE PROFESSOR, can be contacted at AYOUNG@FLUGLE.EDU
NILAND, ASSOCIATE PROFESSOR, can be contacted at MNILAND@FLUGLE.EDU
BATES, ASSISTANT PROFESSOR, can be contacted at JBATES@FLUGLE.EDU
```

5.1.8 Fonctions intégrées

SQL Server fournit un riche ensemble de fonctions intégrées que vous pouvez utiliser pour manipuler et convertir différents types de données. Elles entrent dans les catégories suivantes :

- caractère
- numérique
- date
- conversion
- groupe
- divers

5.2 Spécification d'un critère dans la clause WHERE

5.2.1 Présentation

Généralement, vous ne souhaitez pas récupérer toutes les lignes d'une table, surtout si elle en comporte un grand nombre. SQL fournit une clause WHERE dans laquelle vous pouvez spécifier un critère de recherche à utiliser pour la requête.

Une clause WHERE consiste en une ou plusieurs conditions qui doivent être satisfaites par une ligne pour qu'elle soit extraite par la requête. Si vous souhaitez obtenir la liste des étudiants inscrits en première année (appelés freshmen en anglais) à l'université Flugle, spécifiez le critère **Year = 'FRESHMAN'** dans la clause WHERE.

```
SELECT Last_Name, First_Name, Street_Address
FROM Student
WHERE Year = 'FRESHMAN';
```

```
LAST_NAME                 FIRST_NAME                STREET_ADDRESS
------------------------- ------------------------- --------------
SMYTHE                    JACKSON                   144 OLIVE AVE.
REYNOLDS                  PAULA                     7493 MAPLE ST.
TANAKA                    JEFFREY                   838 PECAN RD.
MALLARD                   HENRY                     123 WALNUT DR.
POSEN                     HUGO                      9100 MAPLE ST.
CLAUSEN                   THOMAS                    901 BIRCH RD.
GOMEZ                     LINDA                     9391 MAPLE ST.
MASSEY                    RICHARD                   431 PINE AVE.
ANASTATIA                 ANNA                      831 BIRCH RD.
HOLMES                    IVAN                      221 OLIVE AVE.
```

5.2.2 Combinaison d'expressions avec AND et OR

Vous pouvez utiliser les opérateurs AND et OR pour combiner plusieurs conditions à satisfaire dans une requête. Pour obtenir la liste des étudiants de deuxième année (appelés sophomore en anglais) dont l'adresse de domicile comporte la chaîne 'ASH', vous procéderiez de la façon suivante :

```
SELECT Last_Name, First_Name, Street_Address
FROM Student
WHERE Year = 'SOPHOMORE' and Street_Address like '%ASH%';
```

```
LAST_NAME                 FIRST_NAME                STREET_ADDRESS
------------------------- ------------------------- -------------------
GORDON                    IVY                       544 ASH ST.
JACKEL                    LINDA                     493 ASH ST.
DEURRE                    PIERRE                    555 ASH ST.
PINKWATER                 PETER                     533 ASH ST.
```

Notez le mot clé LIKE utilisé dans l'exemple précédent. Cet opérateur est l'un des plus puissants en SQL.

La syntaxe de base de l'utilisation de l'opérateur LIKE est la suivante :
nom_colonne LIKE 'critère''

Les variables sont définies ainsi :
- ✓ *nom_colonne* représente une colonne de table valide spécifiée dans la clause FROM.
- ✓ *critère* est une chaîne faisant l'objet de la recherche.

Le caractère pourcentage '%' sert de caractère générique dans ce contexte. Il est équivalent à zéro ou plusieurs caractères. Le caractère de soulignement '_' sert à remplacer n'importe quel caractère unique.

5.3 Tri de données avec la clause ORDER BY

La clause **ORDER BY** désigne les colonnes devant être utilisées pour trier les lignes qui sont retournées par la requête. Cette clause est optionnelle, mais souvenez-vous que l'ordre dans lequel les lignes sont produites par une requête est souvent arbitraire (que la table ait été indexée ou non). Par conséquent, vous spécifierez généralement une clause **ORDER BY** dans une instruction **SELECT.**

Vous souhaitez obtenir la liste des enseignants de l'université Flugle classés d'abord par département, puis par leurs nom et prénom.

```
SELECT Department_ID, Last_Name, First_Name
FROM Instructor
ORDER BY Department_ID, Last_Name, First_Name;
```

```
DEPARTMENT_ID        LAST_NAME                FIRST_NAME
-------------------- ------------------------ -----------
ANTHRO               DANIELS                  LAURA
ANTHRO               HITCHCOCK                BORIS
BIO                  EDWARDS                  SAMANTHA
BIO                  WEISS                    ROBERTA
ECON                 PARKER                   WILLIAM
ENG                  CHU                      STEVEN
ENGL                 CHERNOW                  BESS
HIST                 RESTON                   RAPHAEL
HIST                 TORRES                   PETER
MATH                 CHANG                    ROGER
MATH                 NILAND                   MARTINA
MATH                 YOUNG                    ALLAN
PHILO                ANGELO                   ARTHUR
PHILO                BILLINGS                 BENJAMIN
…
```

Notes :

- Il est possible dans une clause ORDER BY, de spécifier des colonnes qui ne font pas partie de la sélection :
```
SELECT Last_Name
FROM Instructor
ORDER BY Position
```
- Vous pouvez spécifier des colonnes dans cette clause, qu'elles fassent partie ou non d'un index de table.
- Lors de l'utilisation de la clause ORDER BY, par défaut le système classe les lignes par ordre croissant. Pour les classer dans un ordre décroissant, vous devez ajouter le mot clé DESC (pour descending) après le nom de colonne. Vous pouvez spécifier les deux types de classement dans une même clause ORDER BY.

```
SELECT Department_ID, Last_Name, First_Name
FROM Instructor
ORDER BY Department_ID desc, Last_Name, First_Name;
```

```
DEPARTMENT_ID        LAST_NAME                 FIRST_NAME
-------------------- ------------------------- -------------------------
PSYCH                BATES                     JOSEPH
PSYCH                JASON                     JERROLD
PHILO                ANGELO                    ARTHUR
PHILO                BILLINGS                  BENJAMIN
PHILO                RICHARDSON                NANCY
MATH                 CHANG                     ROGER
MATH                 NILAND                    MARTINA
MATH                 YOUNG                     ALLAN
HIST                 RESTON                    RAPHAEL
…
```

5.4 Comptage des lignes d'une table

Si vous souhaitez connaître le nombre de lignes pouvant satisfaire à un critère sans réellement chercher à les extraire, utilisez la fonction **COUNT.** Cette fonction retourne une seule ligne indiquant le nombre de lignes satisfaisant au critère donné.

```
SELECT count(Instructor_ID)
FROM Instructor
WHERE Position = 'ASSOCIATE PROFESSOR';

COUNT(Instructor_ID)
--------------------
                   7
```

Note :
COUNT est une fonction de type groupe. Si on spécifie le nom d'une colonne susceptible de contenir la valeur NULL, le système ne retourne que les lignes contenant une valeur pour la colonne indiquée, c'est-à-dire les lignes non nulles de la colonne.

5.5 Recherche des lignes contenant une valeur NULL

Dans une base relationnelle, une valeur NULL dans une colonne peut notamment avoir la signification suivante : la colonne n'a pas encore reçu de valeur (un étudiant peut provisoirement ne pas avoir de numéro fax).

Si vous souhaitez rechercher les enregistrements d'une table dans lesquels une colonne particulière n'a pas encore reçu de valeur, vous pouvez le spécifier au moyen d'une clause WHERE. Le listing suivant illustre une requête qui extrait la liste des noms d'étudiants ne possédant pas de numéro de fax.

```
SELECT Last_Name, First_Name
FROM Student
WHERE Fax is null;
```

Vous pouvez aussi utiliser l'opérateur NOT pour rechercher les lignes dans lesquelles une valeur de colonne n'est pas NULL. Par exemple, pour rechercher les étudiants possédant un numéro de fax, vous pouvez créer la requête suivante :

```
SELECT Last_Name, First_Name
FROM Student
WHERE Fax is not null;
```

Vous devez savoir comment une valeur NULL est traitée dans les opérations arithmétiques. Pour constater la différence entre cette valeur et une valeur zéro, analysez le listing qui suit. Supposez que vous ayez une table nommée Intelligence possédant deux colonnes : NOM et QI.

Contenu de cette table :

```
SELECT Last_Name, IQ
FROM Intelligence;

LAST_NAME                       IQ
------------------------- ---------
SMITH                          100
GORDON                         125
JONES                          150
WILSON
RICHARDS
```

Vous pouvez constater que Wilson et Richards ont un QI NULL. Si vous souhaitez connaître la moyenne des QI pour l'ensemble des enregistrements de la table, expérimentez la requête suivante :

```
SELECT avg(IQ)
FROM Intelligence;

AVG(IQ)
-------
    125
```

Comme vous pouvez le constater, les lignes contenant la valeur NULL dans la colonne QI n'ont pas été utilisées pour calculer la moyenne. Le système calcule en fait : (100 + 125 + 150) / 3. Si vous changez les valeurs NULL en 0, le résultat serait différent (cf. ci-après).

```
UPDATE Intelligence
SET IQ = 0
WHERE IQ is NULL;

SELECT Last_Name, IQ
FROM Intelligence;

LAST_NAME                       IQ
------------------------- ---------
SMITH                          100
GORDON                         125
JONES                          150
WILSON                           0
RICHARDS                         0

SELECT avg(IQ) FROM Intelligence;

AVG(IQ)
-------
     75
```

5.6 Spécification d'un critère avec l'opérateur LIKE

Vous avez déjà vu un exemple de l'utilisation de l'opérateur LIKE. Les utilisateurs d'un système de gestion de base de données se servent de cet opérateur pour rechercher des lignes lorsqu'ils ne sont pas certains de l'orthographe exacte d'un élément par exemple, cet opérateur peut cependant s'avérer très utile dans bien d'autres cas.

Exemple : sélection des étudiants dont le nom inclut les caractères 'IN'.

```
SELECT Student_ID, Last_Name, First_Name
FROM Student
WHERE Last_Name like '%IN%';

STUDENT_ID           LAST_NAME                FIRST_NAME
-------------------- ------------------------ -----------
10231318             BING                     RUDOLPH
10231304             CHIN                     MICHAEL
10231323             PINKWATER                PETER
```

Exemple : sélection des cours dont la description inclut le mot 'workshop'.

```
SELECT Department_ID, Course_ID, Title
FROM Course
WHERE Description like '%workshop%';

DEPARTMENT_ID        COURSE_ID            TITLE
-------------------- -------------------- --------------------
ECON                 189                  MONETARY POLICY
PHILO                198                  MODERN PHILOSOPHY
ECON                 199                  WORKSHOP ON MARX
HIST                 199                  WORKSHOP ON JEFFERSON
PSYCH                181                  WORKSHOP ON NEUROSES
```

Cependant, si le système de gestion de bases de données utilisé est configuré pour effectuer la distinction de casse, cette requête n'inclura pas les cours dont la description débute par le mot 'Workshop', car la première lettre est une majuscule. Dans un tel cas, il faudrait modifier la requête de la façon suivante :

```
SELECT Department_ID, Course_ID, Title
FROM Course
WHERE upper(Description) like '%WORKSHOP%';

DEPARTMENT_ID        COURSE_ID            TITLE
-------------------- -------------------- -------------------------
ECON                 189                  MONETARY POLICY
PHILO                198                  MODERN PHILOSOPHY
ECON                 199                  WORKSHOP ON MARX
HIST                 199                  WORKSHOP ON JEFFERSON
ENGL                 193                  SEMINAR ON THEME ANALYSIS
PSYCH                181                  WORKSHOP ON NEUROSES
```

Il faut également ajouter qu'il est possible de séparer les chaînes spécifiées dans le critère au moyen du caractère '%'.

```
SELECT Description
FROM Course
WHERE upper(Description) like '%STUDY%BEHAVIOR%';

DESCRIPTION
--------------------------------------------------------------------------------
Discussion and field study of the strange behavior of the Nacimera culture
```

Il peut être nécessaire de rechercher des chaînes comportant le caractère % lui-même. Entre guillemets, il est normalement considéré par le SGBDR comme un caractère générique. Il faut le faire précéder du caractère d'échappement '\' pour qu'il soit interprété littéralement et puisse faire l'objet d'une recherche. Exemple : on souhaite rechercher les descriptions de cours comportant le caractère '%', par conséquent, vous placez '\%' entre deux caractères génériques '%'.

```
SELECT Description
FROM Course
WHERE Description like '%\%%' escape '\';

DESCRIPTION
--------------------------------------------------------
Learn about advanced operations such as % and logarithms.
```

Cette requête utilise le caractère barre oblique arrière '\' pour indiquer au système que le caractère '%' qui suit doit être interprété littéralement. Vous pouvez utiliser la même méthode pour rechercher le caractère de soulignement '_' interprété littéralement et non comme caractère de remplacement.
Voici quelques suggestions d'utilisation de l'opérateur **LIKE** dans vos requêtes :

- Appliquez les fonctions **UPPER** ou **LOWER** aux colonnes sélectionnées afin d'unifier la casse des caractères si cela est nécessaire.
- Si la chaîne recherchée se trouve au début de la valeur de colonne, ne la faites pas précéder d'un caractère générique :

  ```
  SELECT Department_ID, Title_ID, Description
  FROM Course
  WHERE upper(Description) like 'SEMINAR%';
  ```

- De la même manière, si la chaîne recherchée se trouve à la fin de la valeur de colonne, ne la faites pas suivre d'un caractère générique :

  ```
  SELECT Department_ID, Title_ID, Description
  FROM Course
  WHERE upper(Description) like '%TOPICS';
  ```

5.7 Requête avec l'opérateur BETWEEN

Dans ce chapitre, il a été expliqué que l'instruction SELECT peut être construite de plusieurs manières pour parvenir au même résultat. L'opérateur BETWEEN représente un autre bon exemple de cette souplesse.

Cet opérateur est inclusif et fonctionne avec des valeurs numériques, des chaînes et des dates. Par exemple, supposez que le coût actuel d'une unité à l'université Flugle est de 250 dollars. Comme le coût d'un cours est égal à la somme du prix de l'unité et des frais supplémentaires, la liste des cours dont le coût se situe entre 760 et 800 dollars peut être obtenue avec la requête suivante.

```
SELECT Title, Units*250 + Additional_Fees
FROM Course
WHERE (Units*250 + Additional_Fees) between 760 and 800;

TITLE                                UNITS*250+ADDITIONAL_FEES
------------------------------------ -------------------------
ADVANCED ARITHMETIC                                        760
INTRO TO PSYCHOLOGY                                        775
ABNORMAL PSYCHOLOGY                                        770
INTRO TO ECONOMICS                                         775
INTRO TO CIRCUIT THEORY                                    795
INTRO TO DYNAMICS                                          785
```

Le même résultat peut être obtenu avec la requête suivante :

```
SELECT Title, Units*250 + Additional_Fees
FROM Course
WHERE (Units*250 + Additional_Fees)>=760 and (Units*250 +
Additional_Fees)<=800;

TITLE                                UNITS*250+ADDITIONAL_FEES
------------------------------------ ------------------------
ADVANCED ARITHMETIC                                       760
INTRO TO PSYCHOLOGY                                       775
ABNORMAL PSYCHOLOGY                                       770
INTRO TO ECONOMICS                                        775
INTRO TO CIRCUIT THEORY                                   795
INTRO TO DYNAMICS                                         785
```

Comme vous pouvez le constater l'opérateur BETWEEN permet de simplifier la requête.

5.8 Requête avec l'opérateur IN

L'opérateur IN permet de comparer la valeur d'une colonne ou d'une expression avec une liste de valeurs possibles.

La syntaxe d'utilisation est la suivante :

expression **IN** (*expression1*, *expression2*, ... *expressionN*)

Les variables sont définies ainsi :

- *expression* est une expression SQL valide.
- *expression1* à *expressionN* est une liste d'expressions SQL valides.

L'opérateur IN retourne une valeur booléenne qui est VRAI ou FAUX:

- VRAI si l'expression est égale à une des valeurs de la liste des expressions.
- FAUX si l'expression n'est égale à aucune des valeurs de la liste des expressions.

Pour illustrer l'emploi de cet opérateur, supposez que vous souhaitiez rechercher les enseignants appartenant aux départements mathématiques, anthropologie ou psychologie. Si la liste des valeurs possibles est longue, l'opérateur permet de réduire la longueur du programme ainsi que le temps de traitement du gestionnaire de requête SQL. Le listing suivant illustre cette situation.

```
SELECT Last_Name, First_Name, Department_ID
FROM Instructor
WHERE Department_ID in ('MATH','ANTHRO','PSYCH');

LAST_NAME              FIRST_NAME             DEPARTMENT_ID
---------------------- ---------------------- -------------
HITCHCOCK              BORIS                  ANTHRO
DANIELS                LAURA                  ANTHRO
CHANG                  ROGER                  MATH
JASON                  JERROLD                PSYCH
YOUNG                  ALLAN                  MATH
NILAND                 MARTINA                MATH
BATES                  JOSEPH                 PSYCH
```

Vous pouvez combiner le mot clé NOT avec l'opérateur IN pour obtenir l'effet inverse, à savoir que la valeur de retour sera VRAI si l'expression recherchée n'est égale à aucune des expressions données dans la liste.

```
SELECT Last_Name, First_Name, Department_ID
FROM Instructor
WHERE Department_ID not in ('MATH','ANTHRO','PSYCH');

LAST_NAME                 FIRST_NAME                DEPARTMENT_ID
------------------------- ------------------------- -------------
EDWARDS                   SAMANTHA                  BIO
TORRES                    PETER                     HIST
ANGELO                    ARTHUR                    PHILO
RICHARDSON                NANCY                     PHILO
PARKER                    WILLIAM                   ECON
CHERNOW                   BESS                      ENGL
CHU                       STEVEN                    ENG
WEISS                     ROBERTA                   BIO
RESTON                    RAPHAEL                   HIST
BILLINGS                  BENJAMIN                  PHILO
```

5.9 Sélection de colonnes avec un alias

Lorsqu'une sélection comporte une expression complexe, vous pouvez vous y référer au moyen d'un alias que vous lui assignez.

La syntaxe d'utilisation est la suivante :
nom_expression1 [**[AS]** *nom-alias1*], , *nom_expressionN*
[**[AS]** *nom_aliasN*]

Les variables sont définies ainsi :

- *nom_expression* est une expression qui se réfère à un ou plusieurs noms de colonnes.
- *nom_alias* est un alias utilisé pour se référer au *nom_expression* dans d'autres parties de l'instruction **SELECT.**

Notez que le mot clé **as** est optionnel.

Exemple :

```
SELECT Title, Units*250 + Additional_Fees as Course_Cost
FROM Course
WHERE Department_ID = 'ECON';
```

	Title	Course_Cost
1	INTRO TO ECONOMICS	775,00
2	MONETARY POLICY	1500,00
3	WORKSHOP ON MARX	750,00

En assignant l'alias **Course_Cost** à l'expression Units*250 + Additional_Fees, vous bénéficiez de deux avantages :

- Vous disposez d'un nom décrivant exactement l'expression.
- Vous pouvez utiliser l'alias dans une clause ORDER BY.

5.10 Utilisation d'une sous-requête

5.10.1 Introduction

Une sous-requête est définie comme une instruction **SELECT** apparaissant dans d'autres instructions du LMD, à savoir **SELECT, UPDATE, DELETE,** ou **INSERT.** Dans une instruction **SELECT,** une sous-requête fait partie d'une condition dans la clause WHERE. La requête suivante renvoie le titre des cours dont les

frais supplémentaires sont inférieurs ou égaux à la moyenne des frais supplémentaires de l'ensemble des cours :

```
SELECT Title
FROM Course
WHERE Additional_Fees <= (SELECT avg(Additional_Fees) FROM Course);
```

Voici certaines particularités importantes liées à l'utilisation d'une sous-requête :

- La sous-requête doit apparaître entre parenthèses.
- Le nombre de lignes retournées par la sous-requête doit correspondre au nombre de valeurs que la fonction ou l'opérateur attend. Dans l'exemple précédent, l'opérateur <= attend une seule valeur pour sa comparaison, et la fonction **AVG** (qui est une fonction de type groupe) retourne une seule valeur.
- Le nombre de colonnes retournées par la sous-requête doit correspondre au nombre de colonnes que la fonction ou l'opérateur attend.
- La clause ORDER BY n'est pas utilisée au sein d'une sous-requête.

5.10.2 Création d'une nouvelle table avec l'instruction SELECT

Si vous souhaitez effectuer des tests sur le contenu d'une table, ajouter, supprimer, ou modifier des lignes, il serait judicieux de réaliser une copie de la table en question. Imaginez la situation suivante : la table contient une liste de clients sur les dix dernières années, ce qui représente un nombre important de lignes, environ 10 000. Vous souhaitez utiliser un sous-ensemble de ces lignes, en particulier, celles qui concernent les clients demeurant en Californie. Pour cela, vous créez une table en utilisant « SELECT ... INTO ... ».

```
SELECT * INTO Customer_Subset
FROM Customer
WHERE State = 'CA';
```

La table « Customer_Subset » est alors créée.

D'autres systèmes de gestion de bases de données, comme Oracle par exemple, permettent l'utilisation de la syntaxe suivante :

```
CREATE TABLE nom_nouvelle_table
AS
instruction_SELECT
```

Cette syntaxe n'est pas disponible avec SQL Server.

Les variables sont définies ainsi :

- ✓ *nom_nouvelle_table* est le nom de la table à créer.
- ✓ *instruction_SELECT* est une instruction **SELECT** valide.

De plus, ces deux instructions peuvent servir un autre objectif. Si vous souhaitez créer une table possédant la même structure qu'une table existante, avec les mêmes définitions de colonnes, mais vidée de ses données, vous pouvez recourir à la requête suivante :

```
CREATE TABLE nouvelle_table_vide
AS
SELECT *
FROM table-existante
WHERE 1 = 2
```

Vous avez probablement remarqué que 1 n'est jamais égal à 2 😂. C'est la raison pour laquelle aucune ligne de la table existante n'est copiée dans la nouvelle table. Cette dernière possède les mêmes définitions de colonnes, mais aucune donnée. Vous pourriez recourir à n'importe quelle expression fausse pour obtenir ce résultat.

5.11 Résumé

Ce chapitre a introduit des aspects fondamentaux du SQL:

- Le langage de manipulation de données LMD est une catégorie d'instructions SQL qui inclut SELECT, INSERT, UPDATE, et DELETE.
- Une instruction SELECT contient une liste de sélection et une clause FROM qui identifie la table à partir de laquelle les lignes sont extraites.
- Habituellement, l'instruction SELECT inclut une clause WHERE qui spécifie le critère que les lignes retournées doivent satisfaire. La clause ORDER BY est utilisée pour spécifier les colonnes servant à ordonner les lignes retournées par la requête.
- Les résultats renvoyés par chaque instruction SELECT peuvent être considérés comme une table temporaire.
- Spécifiez toujours les colonnes à extraire dans une instruction SELECT.

6 Utilisation des fonctions intégrées existant en SQL

Ce chapitre décrit les instruments offerts par SQL Server pour manipuler des chaînes, des dates et des nombres. Elles vous seront utiles dans les situations suivantes :

- pour convertir les chaînes d'un format à un autre dans une application de base de données;
- pour générer des états;
- pour exécuter des requêtes;
- pour exporter des données d'une base vers une autre.

Le système SQL Server fournit un vaste ensemble de fonctions intégrées et d'opérateurs se destinant à la conversion et à la manipulation de chaînes, de dates et de nombres. Vous constaterez la véritable puissance de ces outils en utilisant des fonctions imbriquées. Ce chapitre présente plusieurs de ces techniques.

6.1 Manipulation de chaînes

Il existe plusieurs fonctions intégrées utiles à la manipulation des chaînes. De nombreux exemples d'utilisation seront présentés dans les pages suivantes.

6.1.1 Recherche de la longueur d'une chaîne

Vous pouvez utiliser la fonction LEN pour trouver la longueur d'une chaîne dans une colonne. Cette fonction renvoie une valeur correspondant au nombre de caractères de l'argument.

```
SELECT Last_Name, len(Last_Name)
FROM Instructor
ORDER BY Last_Name;
```

```
LAST_NAME                 LEN(LAST_NAME)
------------------------- -----------------
ANGELO                                    6
BATES                                     5
BILLINGS                                  8
CHANG                                     5
CHERNOW                                   7
CHU                                       3
DANIELS                                   7
EDWARDS                                   7
HITCHCOCK                                 9
JASON                                     5
NILAND                                    6
PARKER                                    6
POULSON                                   7
……….
```

6.1.2 Extraction d'une portion de chaîne

La fonction SUBSTRING permet d'extraire une portion spécifique d'une chaîne.

Cette fonction est utilisée de la manière suivante :
SUBSTRING (*chaîne*, *position_départ*, *nombre_caractères*)

Les variables sont définies ainsi :

- *chaîne* est une colonne de type caractère ou une expression de type chaîne.
- *position_départ* est la position à partir de laquelle débute l'extraction.
- *nombre_caractères* est le nombre de caractères à renvoyer.

Le listing suivant illustre l'utilisation de la fonction SUBSTRING pour obtenir les quatre premiers caractères du nom de l'enseignant.

```
SELECT Last_Name, substring(Last_Name,1,4)
FROM Instructor
ORDER BY Last_Name;
```

```
ANGELO                    ANGE
BATES                     BATE
BILLINGS                  BILL
CHANG                     CHAN
CHERNOW                   CHER
CHU                       CHU
DANIELS                   DANI
EDWARDS                   EDWA
HITCHCOCK                 HITC
JASON                     JASO
……..
```

L'argument de la fonction SUBSTRING peut être une constante littérale, mais aussi une autre fonction. L'exemple du listing ci-dessous, bien que peu utile, illustre cette possibilité. Supposez que vous souhaitiez extraire les trois derniers caractères du nom d'un enseignant. Tout d'abord, vous utiliseriez la fonction LEN pour trouver la position du dernier caractère. Pour déterminer ensuite la position de départ correcte à passer à la fonction SUBSTRING, vous utiliseriez la formule LEN( ) - (n - 1), où n est le nombre de caractères à extraire. Dans notre exemple : 3 - 1 = 2.

```
SELECT substring(Last_Name,len(Last_Name)-2,3)
FROM Instructor
ORDER BY Last_Name;
```

```
ELO
TES
NGS
ANG
NOW
CHU
ELS
RDS
OCK
SON
……..
```

6.1.3 Recherche d'une portion de chaîne

L'opérateur LIKE a été étudié précédemment. En bref il permet de rechercher dans une expression de type chaîne une suite de caractères spécifiés comme critère. Vous pouvez effectuer des recherches très spécifiques avec cet opérateur au moyen de critères sophistiqués.

Le listing suivant illustre comment utiliser cet opérateur pour extraire toutes les lignes dans lesquelles une description de cours contient la chaîne 'theory' ou 'theories'.

```
SELECT Title, Description
FROM Course
WHERE Description like '%theory%' or Description like '%theories%';
```

```
TITLE                          DESCRIPTION
------------------------------ ----------------------------------------
NUMBER THEORY                  Introduction to number theory;
                               characteristics of natural, rational,
                               and irrational numbers; survey of lucky
                               numbers

INTRO TO PSYCHOLOGY            A general introduction to the theory of
                               human psychology. Topics include
                               Freudian theory, theory of behavior,
                               psychological development of the child,
                               and other topics.

INTRO TO ECONOMICS             A general introduction to the "dismal"
                               science of economics. Topics include
                               supply and demand, the theory of money,
                               theories of production, and other
                               relevant topics.

INTRO TO STRUCTURES            A general introduction to the theory of
                               mechanical structures including levers,
                               gears, beams, and others.

MODERN PHILOSOPHY              A workshop on recent philosophical
                               debates including bioethics,
                               communication theory, social net theory,
                               and philosophical relativism.

INTRO TO CIRCUIT THEORY        A general introduction to the theory of
                               circuits, logical design, boolean
                               theory, and other topics.

INTRO TO DYNAMICS              Introduction to the theory of dynamic
                               systems, both linear and nonlinear.

SEMINAR ON CHAOS               Seminar will explore examples of chaos
                               theory as applied to economic,
                               biological, and physical systems. Class
                               schedule TBD.

EVOLUTIONARY GRAMMAR           Seminar on current trends in English
                               grammar. Topics include apostrophe
                               theory and plurality.

WORKSHOP ON NEUROSES           Intense workshop on the development and
                               perfection of various neuroses.
                               Exploration of neuroses-psychoses
                               transition theories.
```

6.1.4 Remplacement d'une portion de chaîne

Une tâche de manipulation de données courante est la transformation d'une chaîne en une autre, dans une colonne particulière. Supposez que vous souhaitiez modifier la description de cours dans la table Course afin que toutes les descriptions incluant le mot 'seminar' soient modifiées pour lui substituer le mot 'discussion'.

Vous pouvez recourir à la fonction REPLACE qui sert à remplacer une chaîne par une autre dans une colonne.

La syntaxe d'utilisation de cette fonction est :
REPLACE (*chaîne*, *chaîne_existante*, [*chaîne_remplacement*])

Les variables sont définies ainsi :

- *chaîne* est une expression de type chaîne.
- *chaîne_existante* est une chaîne pouvant apparaître dans *chaîne*.
- *chaîne_remplacement* est une chaîne optionnelle par laquelle *chaîne_existante* est remplacée.

Le listing qui suit montre comment changer un titre de cours dans la table Course avec cette fonction. Tout d'abord, une requête permet d'afficher les titres actuels. Une instruction UPDATE est utilisée avec la fonction REPLACE pour substituer le mot 'DISCUSSION' au mot 'SEMINAR'. Une autre requête montre le résultat de la modification.

Si vous ne spécifiez pas de chaîne de substitution dans la fonction REPLACE, la chaîne existante sera supprimée de la colonne.

```
SELECT Title
FROM Course
ORDER BY Title;

TITLE
----------------------------
ABNORMAL PSYCHOLOGY
ADVANCED ARITHMETIC
ANCIENT GREEK PHILOSOPHERS
EARLY AMERICAN HISTORY
EVOLUTIONARY GRAMMAR
GENERAL CALCULUS
INTRO TO ANTHROPOLOGY
INTRO TO BIOLOGY
INTRO TO CIRCUIT THEORY
INTRO TO DYNAMICS
INTRO TO ECONOMICS
INTRO TO ENGLISH LIT
INTRO TO PHILOSOPHY
INTRO TO PSYCHOLOGY
INTRO TO STRUCTURES
INVERTEBRATE ANATOMY
MAMMALIAN ANATOMY
MODERN ENGLISH LIT
ODERN EUROPEAN HISTORY
MODERN PHILOSOPHY
MONETARY POLICY
NUMBER THEORY
PHYS ANTHRO FIELD TECHNIQUES
PRE-CALCULUS
PSYCH IN FILM
```

```
SEMINAR ON CHAOS
SEMINAR ON NACIMERA
SEMINAR ON THEME ANALYSIS
WORKSHOP ON JEFFERSON
WORKSHOP ON MARX
WORKSHOP ON NEUROSES
WORKSHOP ON NORMALITY


UPDATE Course
SET Title = replace(Title,'SEMINAR','DISCUSSION');


SELECT Title
FROM Course
ORDER BY Title;

TITLE
----------------------------
ABNORMAL PSYCHOLOGY
ADVANCED ARITHMETIC
ANCIENT GREEK PHILOSOPHERS
DISCUSSION ON CHAOS
DISCUSSION ON NACIMERA
DISCUSSION ON THEME ANALYSIS
EARLY AMERICAN HISTORY
EVOLUTIONARY GRAMMAR
GENERAL CALCULUS
INTRO TO ANTHROPOLOGY
INTRO TO BIOLOGY
INTRO TO CIRCUIT THEORY
INTRO TO DYNAMICS
INTRO TO ECONOMICS
INTRO TO ENGLISH LIT
INTRO TO PHILOSOPHY
INTRO TO PSYCHOLOGY
INTRO TO STRUCTURES
INVERTEBRATE ANATOMY
MAMMALIAN ANATOMY
MODERN ENGLISH LIT
MODERN EUROPEAN HISTORY
MODERN PHILOSOPHY
MONETARY POLICY
NUMBER THEORY
PHYS ANTHRO FIELD TECHNIQUES
PRE-CALCULUS
PSYCH IN FILM
WORKSHOP ON JEFFERSON
WORKSHOP ON MARX
WORKSHOP ON NEUROSES
WORKSHOP ON NORMALITY
```

6.1.5 Amputation d'une chaîne

Si une colonne de type caractère contient des espaces en début ou en fin de valeur, une recherche basée sur un critère spécifique peut renvoyer un résultat erroné. Pour illustrer ce point, voyez l'exemple du listing suivant. Tout d'abord, une instruction UPDATE est utilisée pour ajouter un espace en début de valeur dans la colonne First_Name. Une requête sur la table Instructor recherche les enseignants dont

le prénom est 'BORIS'. Aucun résultat n'est renvoyé. Une seconde requête recherchant un prénom ' BORIS' (on a ajouté un espace au début) retourne la ligne qui avait été modifiée.

```
UPDATE Instructor
SET First_Name = ' ' + First_Name;

(18 ligne(s) affectée(s))


SELECT Instructor_ID, Last_Name, First_Name
FROM Instructor
WHERE First_Name = 'BORIS';

Aucune ligne n'est sélectionnée.


SELECT Instructor_ID, Last_Name, First_Name
FROM Instructor
WHERE First_Name = ' BORIS';

INSTRUCTOR_ID        LAST_NAME                 FIRST_NAME
-------------------- ------------------------- ----------
M101                 HITCHCOCK                 BORIS
```

Bien sûr, si vous n'aviez pas été au courant de l'espace ajouté en début de valeur, vous auriez été surpris par la réponse.

Pour éviter le retour d'un résultat de requête erroné, supprimez les espaces en début et en fin des lignes insérées ou modifiées dans une table. Les espaces en fin de valeur sont, de loin, les plus courants en comparaison avec ceux situés en début de chaîne.
Le système fournit deux fonctions permettant de supprimer les espaces, LTRIM et RTRIM. La première supprime les espaces en début de chaîne, et la seconde, ceux en fin de chaîne.
Pour supprimer à la fois les espaces en début et en fin de chaîne, imbriquez la fonction LTRIM dans la fonction RTRIM. Si aucun espace n'existe, la chaîne ne sera pas modifiée.

Exemple :

```
UPDATE Instructor
SET First_Name = ltrim(First_Name)
WHERE First_Name like ' %';

(18 ligne(s) affectée(s))
```

Notez que la clause WHERE a été utilisée dans cet exemple pour ne modifier que les lignes dont le prénom est précédé d'un espace. La fonction LTRIM aurait fonctionné sans cette clause. Toutefois, si la table est de grande taille, les performances de l'instruction sont améliorées par l'emploi de cette clause qui réduit le nombre de lignes à celles auxquelles s'applique la fonction.

Comme vous pouvez le constater, LTRIM et RTRIM nécessitent un seul argument, la chaîne à tronquer. Cependant avec certains systèmes de gestion de bases de données, vous pouvez aussi spécifier un second argument optionnel pour les deux fonctions, déterminant un ensemble de caractères à supprimer de la chaîne passée en argument. Oracle par exemple, offre cette fonctionnalité. Ce n'est cependant pas le cas de SQL Server.

Supposez que vous ayez une table nommée Test_Trim avec une seule colonne de type VARCHAR nommée My_Col. Vous savez que certaines lignes débutent par des caractères que vous souhaitez supprimer: 'x', 'y', et 'z'. La fonction LTRIM est utilisée dans une instruction UPDATE à cette fin.

```
SELECT my_col
FROM test_trim;

MY_COL
-------------------------------------------------------------
yzzxHello, world
zyxGoodbye, cruel world
```

Syntaxes applicables avec Oracle.

```
UPDATE test_trim
SET my_col = ltrim(my_col,'xyz');


SELECT my_col FROM test_trim;

MY_COL
-------------------------------------------------------------
Hello, world
Goodbye, cruel world
```

Vous pouvez éliminer d'autres caractères en fin de chaîne selon le même procédé qu'avec la fonction RTRIM.

Note
Les versions actuelles d'Oracle proposent la fonction TRIM permettant d'effectuer des suppressions **à gauche et à droite** dans une chaîne.

6.1.6 Remplissage d'une chaîne

Il existe certaines situations dans lesquelles il est nécessaire de compléter une valeur de chaîne en lui ajoutant des caractères au début ou à la fin. Suivant le système de gestion de bases de données, différentes fonctions sont proposées. Oracle, par exemple, fournit deux fonctions servant cet objectif : LPAD et RPAD. Quant à SQL Server, il s'agit de la fonction REPLICATE.

6.1.6.1 Fonction LPAD (disponible avec Oracle, MySQL)

La syntaxe d'utilisation de la fonction LPAD est :
LPAD (*chaîne*, *n*, *chaîne_remplissage*)

Les variables sont définies ainsi :

- *chaîne* est la constante littérale ou la colonne de type chaîne faisant l'objet du remplissage.
- *n* est la longueur totale de la chaîne retournée par la fonction.
- *chaîne_remplissage* est la chaîne placée au début de chaîne.

Exemple illustrant comment utiliser LPAD pour ajouter des espaces en début de colonne.

```
SELECT lpad(my_col,20) FROM test_trim;

LPAD(MY_COL,20)
--------------------
        Hello, world
Goodbye, cruel world
```

Lorsque l'argument *chaîne_remplissage* n'est pas indiqué, LPAD ajoute par défaut un espace au début de la chaîne. Vous pouvez spécifier une constante littérale que LPAD ajoutera au début de la chaîne. Toutefois, le nombre de caractères qui est ajouté à la chaîne dépend de la valeur de ***n***. Le listing qui suit illustre comment ajouter une valeur fixe à la fonction LENGTH pour servir d'argument à LPAD. En augmentant le nombre ajouté à la valeur retournée par LENGTH, vous pouvez faire débuter plusieurs fois l'argument chaîne avec l'argument *chaîne_remplissage*, comme vous pouvez le voir avec la seconde requête.

```
SELECT lpad(my_col,length(my_col)+8,'You say ') FROM test_trim;

LPAD(MY_COL,LENGTH(MY_COL)+8,'YOUSAY')
--------------------------------------------------------------------------
You say Hello, world
You say Goodbye, cruel world


SELECT lpad(my_col,length(my_col)+16,'You say ') FROM test_trim;

LPAD(MY_COL,LENGTH(MY_COL)+16,'YOUSAY')
--------------------------------------------------------------------------
You say You say Hello, world
You say You say Goodbye, cruel world
```

Vous pouvez ajouter, au début d'une chaîne, le contenu d'une autre colonne. Comme vous le voyez dans la seconde requête du listing suivant, les valeurs de My_Col2 sont ajoutées au début des valeurs de My_Col. Le nombre de fois qu'elles sont ajoutées dépend de la longueur des valeurs de chaque colonne. Si la colonne My_Col est égale à 'California' 1 (10 caractères) et My_Col2 est égale à 'Los Angeles' (11 caractères), LPAD retourne une chaîne dans laquelle la valeur 'Los Angeles' aura été ajoutée, au début de 'California', autant de fois que nécessaire pour remplir 40 caractères, 50 ayant été spécifié comme argument pour ***n***.

Exemple illustrant l'utilisation de LPAD pour concaténer deux colonnes :

```
SELECT * FROM test_trim;

MY_COL                         MY_COL2
------------------------------ -----------
California                     Los Angeles
Michigan                       Jackson
Washington                     Seattle
Oregon                         Portland


SELECT lpad(my_col,50,my_col2) FROM test_trim;

LPAD(MY_COL,50,MY_COL2)
--------------------------------------------------
Los AngelesLos AngelesLos AngelesLos AngCalifornia
JacksonJacksonJacksonJacksonJacksonJacksonMichigan
SeattleSeattleSeattleSeattleSeattleSeattWashington
PortlandPortlandPortlandPortlandPortlandPortOregon
```

En combinant ces fonctions intégrées, vous pouvez construire des expressions élaborées. Le listing qui suit utilise les longueurs de My_Col et My_Col2 comme arguments pour garantir que le remplissage à gauche ne se produit qu'une seule fois. Analysons l'instruction SELECT. Le premier argument de LPAD

est My_Col. Pour le second argument, la longueur de My_Col est ajoutée à celle de My_Col2, plus le nombre 2 pour la chaîne ', ' qui sera placée entre My_Col2 et My_Col. Finalement, le troisième argument de la fonction est la concaténation de la valeur de My_Col2 et ', '.

Exemple illustrant l'utilisation de la fonction LENGTH comme argument de la fonction LPAD :

```
SELECT lpad (my_col,length(my_col)+length(my_col2)+2, my_col2 || ', ')
FROM test_trim;

LPAD(MY_COL,LENGTH(MY_COL)+LENGTH(MY_COL2)+2,MY_COL2||', ')
------------------------------------------------------------
Los Angeles, California
Jackson, Michigan
Seattle, Washington
Portland, Oregon
```

6.1.6.2 Fonction RPAD (disponible avec Oracle, MySQL)

RPAD fonctionne comme LPAD.

Sa syntaxe est la suivante :
RPAD (*chaîne*, *n*, *chaîne_remplissage*)

Les variables sont définies ainsi :

- *chaîne* est la constante littérale ou la colonne de type chaîne faisant l'objet du remplissage.
- *n* est la longueur totale de la chaîne retournée par la fonction.
- *chaîne_remplissage* est la chaîne placée à la fin de *chaîne*.

6.1.6.3 Fonction REPLICATE (disponible avec SQL Server)

Exemple d'utilisation :

```
SELECT replicate('-',5) + ' ID:'
```

	(Aucun nom de colonne)
1	----- ID:

6.1.7 Modification de la casse dans une chaîne

Le système fournit trois fonctions qui permettent de changer la casse des caractères d'une chaîne :

- INITCAP convertit le premier caractère de chaque mot en majuscule, les autres caractères étant retournés en minuscules. **Cette fonction n'est pas disponible avec SQL Server** mais elle est proposée par Oracle.
- LOWER convertit tous les caractères d'une chaîne en minuscules.
- UPPER convertit tous les caractères d'une chaîne en majuscules.

Ces trois fonctions reçoivent un seul argument : la chaîne à manipuler. Le listing ci-dessous illustre l'utilisation de ces fonctions.

```
SELECT lower(Title), upper(Title)
FROM Course
ORDER BY Title;

LOWER(TITLE)                       UPPER(TITLE)
---------------------------------- --------------------------
abnormal psychology                ABNORMAL PSYCHOLOGY
advanced arithmetic                ADVANCED ARITHMETIC
ancient greek philosophers         ANCIENT GREEK PHILOSOPHERS
```

```
early american history              EARLY AMERICAN HISTORY
evolutionary grammar                EVOLUTIONARY GRAMMAR
general calculus                    GENERAL CALCULUS
intro to anthropology               INTRO TO ANTHROPOLOGY
intro to biology                    INTRO TO BIOLOGY
intro to circuit theory             INTRO TO CIRCUIT THEORY
…….
```

L'exemple suivant montre comment la fonction INITCAP convertit tous les caractères en minuscules et met la première lettre de chaque mot en majuscule.

```
SELECT initcap(Title)
FROM Course
ORDER BY Title;
```

Cette fonction n'est pas disponible avec SQL Server.

```
INITCAP(TITLE)
--------------------------
Abnormal Psychology
Advanced Arithmetic
Ancient Greek Philosophers
Early American History
Evolutionary Grammar
General Calculus
Intro To Anthropology
Intro To Biology
Intro To Circuit Theory
Intro To Dynamics
Intro To Economics
Intro To English Lit
……
```

6.1.8 Récupération d'une chaîne avec la fonction DECODE (uniquement disponible avec Oracle)

De nombreuses applications de bases de données se réfèrent à des colonnes qui contiennent des informations codées. Parfois, un concepteur de base de données crée une table pour conserver les codes et leurs descriptions, dans le cas particulier où il s'attend à ce que les codes changent. Dans d'autres situations, la colonne contenant les codes peut apparaître seule sans autre description disponible dans la base de données.

Dans notre exemple de base de données, une table contenant les détails d'un calendrier particulier (dates et durée des cours de la classe) est nécessaire. La table Schedule_Type_Details a donc été créée. Une des colonnes de cette table, Day, conserve les jours de la semaine sous forme d'entiers numériques avec 1 pour Sunday, 2 pour Monday et ainsi de suite.

La plupart des utilisateurs trouveront peu pratique de devoir toujours déchiffrer cette représentation numérique. Heureusement la fonction DECODE est là pour permettre de traduire ce genre de code obscur en constante littérale moins difficile à comprendre.

La syntaxe d'utilisation de cette fonction est :

DECODE (*expression*, *valeur1*, *valeur1_retournée*,...
valeurN, *valeurN_retournée*,
[*valeur_défaut_retournée*])

Les variables sont définies ainsi :

- *expression* est une expression valide.
- *valeurN* est une valeur possible à laquelle *expression* peut être égale.
- *valeurN_retournée* est la valeur retournée par DECODE si *expression* est égale à *valeurN*.
- *valeur_défaut_retournée* est une valeur optionnelle retournée par DECODE si *expression* n'est égale à aucune des valeurs de *valeur1* à *valeurN*.

Le listing qui suit montre comment la fonction DECODE peut traduire la représentation numérique du jour en son nom littéral.

```
SELECT Schedule_ID, Day,decode (Day, 1, 'SUN', 2, 'MON', 3, 'TUE', 4,
'WED', 5, 'THU', 6, 'FRI', 7, 'SAT')
FROM Schedule_Type_Details
ORDER BY Schedule_ID, Day;

SCHEDULE_ID                DAY DEC
-------------------- --------- ---
S180                         6 FRI
T10                          2 MON
T10                          4 WED
T10                          6 FRI
T13                          2 MON
T13                          4 WED
T13                          6 FRI
T15                          2 MON
T15                          4 WED
T15                          6 FRI
TT9                          3 TUE
TT9                          5 THU
```

NB : Cette fonction n'étant pas disponible avec SQL Server, il est possible de créer une fonction personnalisée dans laquelle une structure de type CASE WHEN sera utilisée.

6.1.9 Conversion d'un caractère en sa valeur numérique ASCII

Il est parfois nécessaire d'obtenir le code numérique interne ASCII d'un caractère dans une colonne. La fonction ASCII accomplit cette tâche et attend un seul argument de type chaîne. Cette fonction retourne le code numérique du premier caractère de son argument. L'exemple ci-dessous illustre son utilisation.

```
SELECT Last_Name, ASCII(Last_Name)
FROM Instructor
ORDER BY Last_Name;

LAST_NAME                 ASCII(LAST_NAME)
------------------------- ----------------
ANGELO                                  65
BATES                                   66
BILLINGS                                66
CHANG                                   67
……
```

6.2 Manipulation de données

On constate énormément de différences dans la manière dont les systèmes de bases de données traitent les dates et les heures. SQL Server fournit des types de données spéciaux, DATETIME et SMALLDATETIME, pour gérer les données de type date et heure.

6.2.1 Types de données DATETIME et SMALLDATETIME

Le type de données DATETIME du système SQL Server requiert 8 octets pour le stockage et le type SMALLDATETIME nécessite 4 octets. De plus, lorsque vous définissez une colonne avec ce type, vous pouvez l'utiliser avec toutes les fonctions intégrées manipulant les dates et les heures.

Le type de données « DateTime » permet de stocker des données comprises entre le 01/01/1753 et le 31/12/9999 avec une précision de 3,33 millisecondes.
Le type de données « SmallDateTime » permet de stocker des données comprises entre le 01/01/1900 et le 06/06/2079 avec une précision d'une minute.

6.2.2 Stockage des informations d'heures avec les types DATETIME, SMALLDATETIME

Les types DATETIME et SMALLDATETIME permettent aussi de stocker des informations de temps : heures, minutes et secondes. Vous pouvez utiliser une colonne de type DATETIME (ou SMALLDATETIME) pour stocker uniquement une information de date, une information d'heure, ou les deux.

Si vous choisissez de ne pas employer le type DATETIME (ou SMALLDATETIME) pour stocker de telles informations, vous serez obligé de développer des algorithmes spécifiques pour manipuler les formats que vous avez définis, et vous ne pourrez pas utiliser les fonctions intégrées. Une tâche qui aurait pu être accomplie avec une seule instruction SELECT nécessitera un traitement complémentaire dans un langage de programmation ou un environnement de développement. Le scénario suivant illustre ce qui se produit si vous n'utilisez pas le type de données DATETIME (ou SMALLDATETIME) lorsque son emploi est approprié.

Supposez qu'il vous soit nécessaire de stocker la date d'embauche de chaque enseignant et une date éventuelle de fin d'activité. Si vous avez pris une mauvaise décision et décidé de stocker les dates sous forme numérique dans le format YYMMDD, la date d'embauche pour un employé recruté le 9 mai 1957 équivaudrait à la valeur 570509. Le listing qui suit contient une description de la structure de la table Instructor avec une colonne permettant de stocker la date d'embauche.

```
Name                           Null?    Type
------------------------------ -------- ----
INSTRUCTOR_ID                           VARCHAR(20)
DEPARTMENT_ID                  NOT NULL VARCHAR(20)
LAST_NAME                      NOT NULL VARCHAR(25)
FIRST_NAME                              VARCHAR(25)
MI                                      VARCHAR(1)
POSITION                                VARCHAR(25)
TELEPHONE                               VARCHAR(10)
FAX                                     VARCHAR(10)
EMAIL                                   VARCHAR(100)
HIRE_DATE                               DECIMAL(6)
```

Cette approche pose quelques problèmes, parmi lesquels :

- **Prévision erronée.** Si un enseignant est embauché après 1999, l'instruction SELECT permettant de classer les enseignants en fonction de la date d'embauche ne produirait pas un résultat correct trié. Un enseignant embauché en l'an 2000 apparaîtrait au début de la liste, le format ne stockant que les deux derniers chiffres de l'année d'embauche.
- **Plus de travail, moins de fiabilité.** Comme vous ne pouvez pas vous appuyer sur une validation de date par le système, la charge de développement d'un algorithme de validation de date vous incombe.
- **Fonctionnalité limitée.** Si vous devez effectuer une requête basée sur une date exprimée à l'aide d'une valeur arithmétique, ce qui pourrait être accompli avec une simple instruction SELECT associée à des fonctions intégrées, nécessiterait alors un traitement additionnel complexe. Par exemple, vous ne pourriez pas construire une instruction SELECT qui rechercherait tous les employés dont la date d'embauche serait inférieure à la date moyenne d'embauche.

De plus, l'emploi du type de données DATETIME (ou SMALLDATETIME) permet de garantir la portabilité de vos applications sur d'autres plates-formes.

6.2.3 Obtention de la date et de l'heure courantes avec GETDATE()

Le système SQL Server possède un certain nombre de valeurs appelées pseudo-colonnes (fonctions) auxquelles il est possible de se référer dans des instructions SQL. Une de ces valeurs est GETDATE(). Contrairement à ce que son nom pourrait laisser supposer, elle contient aussi des informations concernant l'heure. C'est une structure très utile pour estampiller les lignes lors des opérations d'insertion ou de modification. De nombreux exemples dans ce chapitre font usage de GETDATE().

6.2.4 Formats de date

Comme cela a déjà été utilisé précédemment, la fonction CONVERT permet de présenter des dates selon le format voulu.

```
SELECT com_ncom, convert(varchar,com_date,103)
FROM tbl_commande_com;
```

Différents codes indiquant le format voulu sont repris dans le tableau qui suit. Les deux colonnes de gauche représentent les valeurs *style* définies pour la conversion de données de type **datetime** ou **smalldatetime** en données de type caractère. Ajoutez 100 à une valeur *style* pour obtenir une année à quatre chiffres incluant le siècle (aaaa).

Sans siècle (aa)	Avec siècle (aaaa)	Standard	Entrée/Sortie (avec siècle)
-	**0** ou **100**	Par défaut	mois jj aaaa hh:miAM (ou PM)
1	**101**	États-Unis	mm/jj/aaaa
2	**102**	ANSI	aaaa.mm.jj
3	**103**	Anglais/Français	jj/mm/aaaa
4	**104**	Allemand	jj.mm.aaaa
5	**105**	Italien	jj-mm-aaaa
6	**106**	-	jj mois aaaa
7	**107**	-	mois jj, aaaa
8	**108**	-	hh:mi:ss
-	**9** ou **109**	Valeur par défaut + millièmes de secondes	mois jj aaaa hh:mi:ss:mmmAM (ou PM)
10	**110**	USA	mm-jj-aaaa
11	**111**	Japon	aaaa/mm/jj
12	**112**	ISO	aaaammjj
-	**13** ou **113**	Valeur par défaut Europe + millièmes de secondes	jj mois en abrégé aaaa hh:mi:ss:mmm (24h)

6.3 Déclaration d'une colonne numérique

6.3.1 Présentation

Comme vous l'avez déjà vu, un type de colonne est spécifié dans les instructions CREATE TABLE et ALTER TABLE.
La syntaxe généralement utilisée pour déclarer un type de données numérique est :
DECIMAL ([*précision* [, *échelle*]])

Les variables sont définies ainsi :

- *précision* est un argument optionnel qui spécifie le nombre de chiffres fixant la précision avec laquelle le système doit stocker les valeurs de colonne.
- *échelle* est un argument optionnel qui indique le nombre de chiffres situés à droite de la virgule que le système doit stocker.

Si vous ne spécifiez pas de précision ou d'échelle, le système accepte une précision allant au maximum jusqu'à 38 chiffres. Lorsque vous spécifiez une colonne, il est conseillé de limiter la largeur en utilisant

une précision appropriée pour stocker les valeurs numériques. Par exemple, si une colonne est prévue pour stocker la température du corps en degrés Celsius, vous la déclarez de la manière suivante :

```
Body_Temp_C Decimal(3,1)
```

```
Une précision de 3 et une échelle de 1 permet à la colonne de stocker un
total de 3 chiffres, dont un chiffre à droite de la virgule. Le listing
suivant illustre un exemple d'insertion d'une valeur respectant la préci-
sion et l'échelle prévues.
```

```
UPDATE Patient
SET Body_Temp_Deg_C = 37.2
WHERE Patient_ID = 'A2002';

(1 ligne(s) affectée(s))
```

```
Le listing qui suit illustre un échec d'insertion d'un nombre ne respectant
pas la précision et l'échelle déclarées.
```

```
UPDATE Patient
SET Body_Temp_Deg_C = 127.3
WHERE Patient_ID = 'A2002';

Msg 8115, Niveau 16, État 8, Ligne 1
```

Une erreur de dépassement arithmétique s'est produite lors de la conversion de numeric en type de données numeric.

L'instruction a été arrêtée.

6.3.2 Utilisation de fonctions intégrées pour les statistiques

SQL Server fournit différentes fonctions statistiques parmi lesquelles nous pouvons citer :

- AVG (*value*), calcule la moyenne de son argument pour l'ensemble des lignes auxquelles elle s'applique.
- STDEV (*value*), retourne la déviation standard de son argument pour l'ensemble des lignes auxquelles elle s'applique.
- VAR (*value*), retourne la variance de son argument pour l'ensemble des lignes auxquelles elle s'applique.

L'utilisation de ces fonctions de groupe est décrite en détail dans le chapitre suivant.

6.3.3 Nombres arrondis et tronqués explicitement

SQL Server fournit différentes fonctions intégrées servant à arrondir et à tronquer des nombres décimaux.

ROUND(*valeur*, *échelle*)
FLOOR(*valeur*)
CEILING(*valeur*)

Dans ces fonctions, *valeur* est une expression numérique et *échelle*, un argument optionnel indiquant le nombre de chiffres décimaux que la fonction devrait utiliser lors du calcul de l'arrondi ou de la troncation, la valeur par défaut étant 0.

Les exemples suivants illustrent l'utilisation de ces fonctions.

6.3.3.1 Fonction ROUND

Cette fonction reçoit deux arguments : une expression numérique et un nombre correspondant au nombre de chiffres à utiliser lors du calcul de l'arrondi. La valeur sera arrondie au nombre décimal le plus proche avec, à droite de la virgule, le nombre de chiffres spécifié. Cette fonction peut être employée avec des constantes numériques (voir exemple ci-après).

```
SELECT ROUND(123.2,0);

ROUND(123.2,0)
------------
       123.0


SELECT ROUND(123.27,1);

ROUND(123.27,1)
---------------
         123.30


SELECT ROUND(101.8,0);

ROUND(101.8,0)
------------
       102.0
```

6.3.3.2 Fonction FLOOR

Cette fonction est presque identique à la fonction TRUNC, à cette différence près qu'elle ne peut pas tronquer un nombre décimal. Elle retourne l'entier qui est inférieur ou égal à son argument numérique.

```
SELECT FLOOR(128.3);

FLOOR(128.3)

------------

         128
```

6.3.3.3 Fonction CEILING

Cette fonction retourne l'entier immédiatement supérieur ou égal à son argument numérique. Elle peut accepter des constantes.

```
SELECT CEILING(101.8)

CEILING(101.8)
-------------
          102
```

6.3.4 Recherche des valeurs extrêmes

Vous pouvez utiliser les fonctions MAX et MIN pour rechercher la valeur la plus grande ou la plus petite dans une colonne particulière. Techniquement, ce sont des fonctions de groupe. Toutefois, vous n'avez pas besoin de spécifier la clause GROUP BY dans une instruction SELECT pour faire appel à ces

fonctions. Le listing suivant montre comment récupérer les estimations, la plus grande et la plus petite, de frais additionnels dans la table Course.

```
SELECT MIN(Additional_Fees), MAX(Additional_Fees)
FROM Course;

MIN(ADDITIONAL_FEES) MAX(ADDITIONAL_FEES)
-------------------- --------------------
                   0                  750
```

6.3.5 Déterminer si une valeur est NULL

Lors du développement d'une application accédant à une base SQL Server, vous pouvez rencontrer des situations dans lesquelles un écran ou un état produira comme réponse à une recherche sur une colonne la valeur NULL. Si vous souhaitez obtenir dans ce cas une valeur quelconque de substitution, vous pouvez employer la fonction COALESCE pour effectuer le remplacement.

La syntaxe de cette fonction est :

COALESCE (*valeur_colonne*, *valeur_substitution*)

Les variables sont définies ainsi :
- ✓ *valeur_colonne* est la valeur de colonne à évaluer.
- ✓ *colonne_substitution* est la valeur que la fonction retournera si *valeur_colonne* est NULL.

Le listing suivant en donne un exemple. Imaginez que la table Course soit modifiée afin que la colonne Additional_Fees reçoive la valeur NULL lorsque les frais sont égaux à zéro. Vous interrogez ensuite la table et constatez que l'absence de frais concerne beaucoup de lignes dans cette colonne. Lors de l'affichage, vous utilisez la fonction COALESCE pour retourner un zéro lorsque la valeur de frais est NULL.

```
UPDATE course
SET additional_fees = null
WHERE additional_fees = 0;


SELECT Department_ID, Course_ID, Additional_Fees
FROM Course
ORDER BY Department_ID, Course_ID;

DEPART COURS ADDITIONAL_FEES
------ ----- ---------------
ANTHRO 101
ANTHRO 174                55
ANTHRO 189               7.5
BIO    101                55
BIO    177                65
BIO    178                70
ECON   101                25
ECON   189               750
ECON   199
ENG    101                75
ENG    102                45
ENG    103                35
ENG    199                45
ENGL   101
ENGL   189
ENGL   192
```

```
ENGL   193
IST    115
HIST   184
HIST   199
MATH   101
MATH   189
MATH   50
ATH    51                  10
PHILO  101
PHILO  174
PHILO  198
PSYCH  101                 25
PSYCH  181                 75
PSYCH  183                 45
PSYCH  185                 20
PSYCH  501                 75
```

```
SELECT Department_ID, Course_ID, COALESCE(Additional_Fees,0)
FROM Course
ORDER BY Department_ID, Course_ID;

DEPART COURS COALESCE(ADDITIONAL_FEES,0)
------ ----- --------------------------
ANTHRO 101                            0
ANTHRO 174                           55
ANTHRO 189                          7.5
BIO    101                           55
BIO    177                           65
BIO    178                           70
CON   101                            25
ECON   189                          750
ECON   199                            0
ENG    101                           75
ENG    102                           45
ENG    103                           35
ENG    199                           45
ENGL   101                            0
ENGL   189                            0
……
```

6.4 Résumé

Ce chapitre a présenté une variété de fonctions intégrées et d'opérateurs, utilisés pour manipuler des valeurs de type chaîne, numérique, date et heure. Parmi ces fonctions, vous avez vu :

- SUBSTRING qui retourne une portion spécifique de chaîne.
- LEN qui retourne la longueur d'une expression chaîne.
- LPAD et RPAD qui placent des caractères de remplissage respectivement à gauche et à droite d'une chaîne (ces fonctions ne sont pas disponibles avec tous les SGBDR).
- LTRIM et RTRIM qui suppriment les espaces ou d'autres caractères, respectivement à gauche et à droite d'une chaîne.
- REPLACE qui substitue une chaîne à une autre.
- LOWER, UPPER qui contrôlent l'utilisation de majuscules et de minuscules dans une chaîne.
- DECODE qui traduit des valeurs de colonne.
- ASCII qui retourne le code interne numérique ASCII correspondant à un caractère.

De plus, vous avez étudié plusieurs aspects liés à l'utilisation des valeurs de date et d'heure :

- Les types de données DATETIME et SMALLDATETIME servent à conserver dans un même espace de stockage des informations de date et d'heure.
- Les types de données DATETIME et SMALLDATETIME seront utilisés lorsque c'est possible, même lorsqu'il s'agit de ne stocker qu'une donnée de date ou une donnée d'heure.
- GETDATE() est une fonction qui retourne toujours la date et l'heure courantes.

Gardez à l'esprit les informations suivantes lorsque vous formez des instructions SQL pour votre application :

- Spécifiez les colonnes numériques avec le type de données DECIMAL. D'autres types existent : TINYINT, INT, ...
- MAX et MIN sont des fonctions de groupe qui retournent respectivement la valeur maximale et minimale d'un ensemble de lignes d'une colonne ou d'une expression.
- La fonction COALESCE retourne une valeur différente, telle que 0 à la place d'une valeur NULL.

7 Requêtes SQL avancées

Vous avez déjà appris à rechercher des lignes avec une instruction SELECT appliquée à une seule table. Cette instruction possède une grande souplesse d'emploi et l'opération d'extraction n'est qu'une de ses nombreuses fonctionnalités. Ce chapitre vous permettra de découvrir la grande puissance de cette instruction en présentant ses fonctionnalités avancées, les clauses GROUP BY et HAVING ainsi que l'opération de jointure.

La syntaxe de l'instruction SELECT n'est pas toujours facile à déchiffrer. Vous ne pouvez pas réellement comprendre l'utilisation de ses clauses au seul vu des descriptions de sa syntaxe présentées dans la documentation de SQL Server. Par conséquent, étudiez les nombreux exemples de ce chapitre qui vous montrent ce qu'il est permis de faire et ce qui doit être évité.

7.1 Fonctions de groupe

7.1.1 Introduction

Le premier sujet d'étude porte sur les fonctions intégrées de SQL Server qui opèrent sur les groupes de lignes. Autant vous le dire tout de suite, cette discussion constitue un bon exemple de la singularité des caractéristiques de SQL. Même s'il s'agit de fonctions de groupe, elles ne nécessitent pas l'emploi de la clause GROUP BY :

- COUNT compte le nombre de lignes extraites par le système qui répondent au critère de recherche.
- MAX et MIN retournent respectivement la valeur maximale et minimale de la colonne spécifiée.
- AVG et SUM calculent respectivement la moyenne et la somme des valeurs d'une colonne.
- STDEV et VARIANCE calculent respectivement la déviation et la variance standard d'une colonne.

Chacune de ces fonctions retourne une simple valeur. Voici quelques exemples détaillés.

7.1.2 Fonction COUNT

Cette fonction comporte deux versions : COUNT(*) qui compte toutes les lignes d'une table qui répondent à un critère défini, et COUNT(*nom_colonne*) qui compte toutes les lignes d'une table répondant à un critère défini et possédant une valeur non nulle dans *nom_colonne*.
Pour illustrer l'utilisation de COUNT(*), le listing suivant montre comment vous compteriez les cours proposés par le département de biologie.

```
SELECT count(*)
FROM Course
WHERE Department_ID = 'BIO';

COUNT(*)
---------
        3
```

Pour compter le nombre d'étudiants possédant un numéro de fax, il faut donner la colonne Fax en argument de la fonction.

```
SELECT count(Fax)
FROM Student;

COUNT(FAX)
----------
         3
```

7.1.3 Fonctions MAX et MIN

Vous pouvez combiner les fonctions MAX et MIN dans une seule instruction SELECT. Le listing suivant illustre l'emploi de ces fonctions. La première requête applique les fonctions à une colonne numérique, Additional_Fees; et la seconde, à une colonne caractère, Instructor_ID.

```
SELECT MAX(Additional_Fees), MIN(Additional_Fees)
FROM Course;

MAX(ADDITIONAL_FEES) MIN(ADDITIONAL_FEES)
-------------------- --------------------
                 750                    0

SELECT MAX(Instructor_ID), MIN(Instructor_ID)
FROM Instructor;

MAX(INSTRUCTOR_ID)   MIN(INSTRUCTOR_ID)
-------------------- --------------------
Y561                 A612
```

7.1.4 Fonctions AVG et SUM

Ces fonctions opèrent de façon similaire aux fonctions MIN et MAX (voir listing ci-après). Une requête sur la table Course recherche la moyenne et le total des frais additionnels pour un cours.

```
SELECT AVG(Additional_Fees), SUM(Additional_Fees)
FROM Course;

AVG(ADDITIONAL_FEES) SUM(ADDITIONAL_FEES)
-------------------- --------------------
           46.171875                1477.5
```

Notez que la fonction AVG occulte les valeurs NULL dans son calcul.

7.1.5 Combinaison de colonnes et de fonctions de groupe

Vous ne pouvez associer des colonnes et des fonctions de groupe dans la liste de sélection d'une instruction SELECT que par le biais de la clause GROUP BY.
Le listing ci-après fournit un exemple de requête dans laquelle deux éléments composent la liste de sélection, Department_ID et SUM(Additional_Fees). Toutefois, pour cette requête, le système retourne un code d'erreur. Pour comprendre pourquoi, analysons la sélection : une fonction de groupe ne retourne qu'une seule ligne, tandis que Department_ID retourne autant de lignes qu'il en existe dans la table. Comme il n'est pas logique de retourner ces deux types de valeurs en même temps, le système produit à la place un message d'erreur. Toutefois, en utilisant une clause GROUP BY, vous pouvez effectuer ce genre de combinaison dans la liste de sélection. Les prochaines sections étudieront plus en détail l'emploi de cette clause.

```
SELECT Department_ID, SUM(Additional_Fees)
FROM Course;

Msg 8120, Niveau 16, État 1, Ligne 1
La colonne 'Course.Department_ID' n'est pas valide dans la liste de sélection parce qu'elle n'est pas contenue dans une fonction d'agrégation ou dans la clause GROUP BY.
```

7.2 Recherche de lignes distinctes

L'instruction SELECT possède un mot clé optionnel que nous n'avons pas rencontré jusqu'ici, DISTINCT. Ce mot clé suit l'ordre SELECT et indique au système de ne retourner pour les colonnes spécifiées que les valeurs distinctes. Le listing suivant montre une requête permettant d'obtenir une liste des villes de résidence des étudiants.

```
SELECT DISTINCT City
FROM Student;

CITY
-------------------------
DOVER
SPRINGFIELD
```

L'option DISTINCT est très utile pour rechercher l'ensemble des valeurs distinctes d'une colonne. Elle propose une méthode permettant de déterminer rapidement comment les valeurs sont groupées dans une table. Si vous ne spécifiez pas cette option, le système extrait toutes les lignes qui satisfont au critère de la clause WHERE. Par défaut, il utilise l'option ALL pour extraire toutes les lignes. Les deux instructions suivantes sont équivalentes :

```
SELECT City FROM Student;
```

et

```
SELECT ALL City FROM Student;
```

7.3 Groupement de lignes – Les clauses GROUP BY et HAVING

La clause GROUP BY est une autre partie de l'instruction SELECT. Cette clause optionnelle indique au système de grouper les lignes selon les valeurs distinctes qui existent pour les colonnes spécifiées. De plus, la clause HAVING peut être combinée avec la clause GROUP BY pour mieux cibler les lignes recherchées.

Même si les exemples sont plus parlants, voici tout d'abord une brève description de sa syntaxe :

```
SELECT liste_sélection
FROM liste_table
[WHERE
condition [AND | OR]
…
condition]
[GROUP BY colonne1, colonne2,.... colonneN]
[HAVING condition]
[ORDER BY colonne1, colonne2,...
```

Les variables sont définies ainsi :

- *liste_sélection* est un ensemble de colonnes et d'expressions relatives aux tables listées dans *liste_table*.
- *liste_table* énumère les tables à partir desquelles les lignes sont extraites.
- *condition* est une condition SQL valide.
- *colonne1* à *colonneN* représentent les colonnes contenues dans *liste_table*.

Note :
Lors de la création d'une instruction SELECT composée de nombreux éléments de syntaxe (colonnes, expressions, tables et conditions), il est courant d'introduire des erreurs qui entraînent le rejet de la requête par le système. Il arrive également que des requêtes soient traitées sans erreur par le système, mais fournissent des résultats difficiles à interpréter. Lorsque ces situations se produisent, la meilleure solution est de revenir à une requête de base. Analysez-la, et si elle ne vous paraît plus très claire, décomposer la puis reconstituer la, élément par élément, jusqu'à obtenir le résultat dans la forme désirée.

Examinez la manière dont vous avez utilisé les fonctions de groupe, les clauses GROUP BY et HAVING, ainsi que toute expression conditionnelle de jointure.

Le listing suivant donne un exemple de requête qui récupère une liste des identifiants de département conservés dans la colonne Department_ID de la table Course.

```
SELECT Department_ID
FROM Course
GROUP BY Department_ID;

DEPARTMENT_ID
--------------------
ANTHRO
BIO
ECON
ENG
ENGL
HIST
MATH
PHILO
PSYCH
```

L'exemple qui suit illustre comment compter le nombre de cours associés à chaque département.

```
SELECT Department_ID, count(*)
FROM Course
GROUP BY Department_ID;

DEPARTMENT_ID         COUNT(*)
-------------------- ---------
ANTHRO                       3
BIO                          3
ECON                         3
ENG                          4
ENGL                         4
HIST                         3
MATH                         4
PHILO                        3
PSYCH                        5
```

Le listing suivant illustre l'utilisation conjointe des clauses GROUP BY et HAVING. La clause HAVING peut servir à rechercher les départements proposant 4 cours.

```
SELECT Department_ID, count(*)
FROM Course
GROUP BY Department_ID
HAVING count(*) = 4;

DEPARTMENT_ID         COUNT(*)
-------------------- ---------
ENG                          4
ENGL                         4
MATH                         4
```

Les fonctions de groupe ne peuvent pas être combinées avec des colonnes dans la liste de sélection à moins d'utiliser la clause GROUP BY. Par exemple, vous pouvez obtenir la valeur moyenne des frais additionnels par département.

```
SELECT Department_ID, avg(Additional_Fees)
FROM Course
GROUP BY Department_ID;

DEPARTMENT_ID        AVG(ADDITIONAL_FEES)
-------------------- --------------------
ANTHRO                           20.833333
BIO                              63.333333
ECON                             258.33333
ENG                                     50
ENGL                                     0
HIST                                     0
MATH                                   2.5
PHILO                                    0
PSYCH                                   48
```

Le prochain exemple met en jeu une autre base de données, celle du système d'information gérant les patients d'un hôpital. Les clauses HAVING et GROUP BY sont utilisées pour l'identification des lignes dupliquées dans une table. Il est possible parfois que vous ne puissiez pas ajouter de clé primaire à une table, car la colonne choisie contient des valeurs en double (voir listing ci-après). Dans cet exemple, une requête utilise ces deux clauses pour identifier les valeurs dupliquées dans la colonne Patient_ID, et une autre requête permet de déterminer les lignes à supprimer. Après la suppression des lignes, la clé primaire peut être intégrée à la table.

Listing permettant d'identifier les lignes dupliquées :

```
ALTER TABLE Patient
ADD CONSTRAINT Patient_PK PRIMARY KEY (Patient_ID);

Msg 1505, Niveau 16, État 1, Ligne 1
L'instruction CREATE UNIQUE INDEX a été interrompue, car une clé dupliquée a été
trouvée pour l'objet 'dbo.Patient' et l'index 'Patient_PK'. Valeur de clé dupli-
quée : (A2002).
Msg 1750, Niveau 16, État 0, Ligne 1
Impossible de créer la contrainte. Voir les erreurs précédentes.
L'instruction a été arrêtée.

SELECT Patient_ID
FROM Patient
GROUP BY Patient_ID
HAVING COUNT(*) > 1;

PATIENT
-------
GG9999

SELECT *
FROM Patient
WHERE Patient_ID = 'GG9999';

PATIENT   BODY_TEMP_DEG_C    FEVER_CLASS
------    ---------------    -------------
GG9999             43.6      LETHAL FEVER
GG9999             40

DELETE FROM Patient
WHERE Patient_ID = 'GG9999' and BODY_TEMP_DEG_C = 40;

ALTER TABLE Patient
ADD CONSTRAINT Patient_PK PRIMARY KEY (Patient_ID);

Commande(s) réussie(s).
```

7.4 Gestion d'informations hiérarchiques

Ce qui est présenté dans ce point n'est applicable qu'avec Oracle pas avec SQL Server.

7.4.1 Introduction

Un exemple de données hiérarchiques pourrait être une nomenclature décrivant toutes les parties, les sous-assemblages et assemblages composant un produit fini. Un constructeur d'avions peut disposer d'une nomenclature composée de milliers de niveaux pour classer des millions de pièces.

Des données hiérarchiques sont des informations contenues dans une table possédant au minimum une colonne spécifiant la clé parent ou la clé de niveau supérieur, et une autre colonne définissant la clé primaire de la ligne courante.

Les bases de données relationnelles en général et SQL en particulier gèrent-ils ce type de hiérarchie ? La réponse est nuancée. La bonne nouvelle est que SQL fournit un support pour ce genre d'organisation de données par l'intermédiaire de la clause CONNECT BY. La mauvaise nouvelle est que ce support est limité et que la syntaxe n'est pas intuitive. Néanmoins, voici un exemple de l'utilisation de SQL pour naviguer à travers une organisation hiérarchique.

Une table composée de trois colonnes nous servira à illustrer l'emploi de la clause CONNECT BY.

- Product_ID Le produit.
- Assembly_ID L'assemblage ou le sous-assemblage auquel l'identifiant de produit appartient.
- Description La description du produit.

Ce schéma de table se base sur l'idée que chaque produit appartient soit au produit fini, soit à un assemblage, excepté pour le produit fini lui-même. Les lignes de la table Product_Assembly illustrent cette idée. Une caméra, la X1000, est composée de plusieurs pièces et d'un sous-assemblage, B200. Le listing qui suit affiche le contenu de la table Product_Assembly.

```
SELECT Assembly_ID, Product_ID, Description
FROM Product_Assembly
ORDER BY Assembly_ID, Product_ID;

ASSEMBLY_ID          PRODUCT_ID   DESCRIPTION
------------ ------------ ------------------------------------------
B200         I101         Titanium alloy, teflon-coated iris
B200         S42          Variable-speed shutter, standard
B200         W123         Manual film advance, winder
X1000        B200         Black body, stainless steel frame camera body
X1000        F100         Blue filter - standard
X1000        F55          Xenon flash unit
X1000        L100         100mm lens - standard
X1000        S04          4 foot camera strap, leather
             X1000        Complete camera

9 rows selected.
```

Vu la faible quantité de données présentées, il est aisé de conceptualiser l'organisation des pièces. Cet exemple nous permettra toutefois d'illustrer l'organisation hiérarchique.

Le listing suivant fournit un exemple de la manière dont les données hiérarchiques peuvent être extraites au moyen d'une instruction SELECT. Celle-ci est accompagnée de deux nouvelles clauses : START WITH et CONNECT BY. La première identifie la ligne de niveau supérieur pour laquelle le système devrait rechercher les lignes subordonnées - vous recherchez les lignes subordonnées à X1000. La seconde clause indique au système comment présenter les lignes. L'opérateur PRIOR demande une

présentation des lignes de telle sorte que chaque identifiant d'assemblage d'une ligne enfant est égal à l'identifiant de produit de sa ligne parent.

```
SELECT lpad(' ',2*(level-1)) || Assembly_ID Assembly_ID,
Product_ID, Description
FROM Product_Assembly
START WITH Product_ID = 'X1000'
CONNECT BY prior Product_ID = Assembly_ID;

ASSEMBLY_ID  PRODUCT_ID   DESCRIPTION
------------ ------------ ----------------------------------------
             X1000        Complete camera
X1000        L100         100mm lens - standard
X1000        B200         Black body, stainless steel frame camera body
B200         S42          Variable-speed shutter, standard
B200         I101         Titanium alloy, teflon-coated iris
B200         W123         Manual film advance, winder
X1000        S04          4 foot camera strap, leather
X1000        F100         Blue filter - standard
X1000        F55          Xenon flash unit

9 rows selected.
```

Afin d'indenter chaque ligne de façon à refléter son niveau hiérarchique, LEVEL a été intégré dans la fonction LPAD de sorte à faire correspondre le nombre d'espaces blancs qu'elle renvoie avec le niveau hiérarchique (LEVEL est une pseudo-colonne qui retourne le niveau hiérarchique de chaque ligne, de 1 pour le niveau le plus élevé à N pour le niveau le plus détaillé). Le résultat de LPAD est concaténé avec l'identifiant d'assemblage Assembly_ID.

Pour consulter un sous-assemblage particulier, spécifiez l'identifiant d'assemblage Assembly_ID dans la clause START WITH. L'instruction SELECT du listing suivant illustre cette procédure. Elle montre aussi la valeur de LEVEL pour chaque ligne.

```
SQL> SELECT lpad(' ',2*(level-1)) || Assembly_ID Assembly_ID,
2  Product_ID, Level, Description
3  FROM Product_Assembly
4  start with Product_ID = 'B200'
5  connect by prior Product_ID = Assembly_ID;

ASSEMBLY_ID     PRODUCT_ID  LEVEL   DESCRIPTION
--------------- ----------------   ---------------------------------
X1000           B200            1  Black body, stainless steel frame camera body
B200            S42             2  Variable-speed shutter, standard
B200            I101            2  Titanium alloy, teflon-coated iris
B200            W123            2  Manual film advance, winder
```

Certains problèmes sont à signaler concernant l'emploi d'une instruction SELECT avec un schéma hiérarchique. Tout d'abord, une telle instruction ne peut pas être utilisée pour joindre deux ou plusieurs tables (sujet traité plus loin dans ce chapitre). Ensuite, si vous spécifiez une clause ORDER BY, vous détruisez le classement hiérarchique des lignes retournées par la requête.

Notes :

À faire :

Utiliser une structure hiérarchique pour une table qui contient des informations hiérarchiques sur plusieurs niveaux. La structure devrait inclure une colonne clé parent.

À ne pas faire :

Utiliser une structure hiérarchique alors que les informations modélisées ne forment pas une hiérarchie à plusieurs niveaux.

7.4.2 L'opérateur EXISTS

En complément aux opérateurs SQL déjà étudiés, vous devez vous familiariser avec l'opérateur EXISTS. Il opère sur une sous-requête et retourne une des valeurs booléennes suivantes :

- TRUE si la sous-requête retourne au moins une ligne;
- FALSE si aucune ligne n'est renvoyée par la sous-requête.

Observez l'exemple suivant portant sur les tables Class et Instructor. Imaginez que vous souhaitiez déterminer quels enseignants ont été prévus pour dispenser un cours. Vous soumettez une requête en utilisant l'opérateur EXISTS.

```
SELECT Last_Name, First_Name, Position
FROM Instructor I
WHERE exists  (SELECT * FROM Class C
   WHERE I.Instructor_ID = C.Instructor_ID);
```

```
LAST_NAME                 FIRST_NAME                POSITION
------------------------- ------------------------- --------------------
CHANG                     ROGER                     ASSISTANT PROFESSOR
JASON                     JERROLD                   ASSOCIATE PROFESSOR
TORRES                    PETER                     PROFESSOR
RICHARDSON                NANCY                     ASSISTANT PROFESSOR
PARKER                    WILLIAM                   PROFESSOR
…
```

L'opérateur EXISTS est utile dans les situations où vous n'êtes pas intéressé par les valeurs de colonnes retournées par la sous-requête. Notez la manière dont les alias, C pour Class et I pour Instructor, ont été employés pour abréger la frappe et améliorer la lisibilité de la requête. Vous pouvez aussi faire précéder cet opérateur de la négation NOT. Le listing qui suit illustre la façon dont vous pouvez obtenir la liste des enseignants non prévus pour un cours particulier.

NB : Un alias est un nom de remplacement pour une table, une colonne ou une expression utilisée dans une requête. L'alias n'existe que pour la durée de la requête. Il sert à la simplifier ou à la clarifier.

```
SELECT Last_Name, First_Name, Position
FROM Instructor I
WHERE
not exists (SELECT * FROM Class C
WHERE I.Instructor_ID = C.Instructor_ID);
```

```
LAST_NAME                 FIRST_NAME                POSITION
------------------------- ------------------------- --------------------
HITCHCOCK                 BORIS                     PROFESSOR
DANIELS                   LAURA                     ASSOCIATE PROFESSOR
EDWARDS                   SAMANTHA                  PROFESSOR
ANGELO                    ARTHUR                    ASSOCIATE PROFESSOR
…
```

Comme vous pouvez le constater à la ligne 5, une sous-requête récupère les colonnes de la table Class pour chaque classe assurée par l'enseignant identifié par Instructor_ID. L'opérateur NOT EXISTS empêchera la requête de renvoyer les valeurs des trois colonnes Last_Name, First_Name et Position pour les lignes que la sous-requête produirait.

7.5 Opérations de jointure

7.5.1 Présentation

À présent que vous détenez une certaine connaissance de SQL, vous êtes prêt à aborder les jointures. Une jointure sert à rechercher des lignes à partir de deux ou plusieurs tables disposant d'un ensemble de valeurs communes. Une base de données relationnelle serait fortement démunie sans cette possibilité de créer des jointures.

L'opération de jointure est un mécanisme permettant de créer des associations **afin d'obtenir des résultats basés sur des valeurs provenant de deux ou plusieurs tables**. On utilise pour cela un critère de recherche dans une clause WHERE, une condition vérifiant qu'une colonne (ou plusieurs) d'une des tables mises en relation - la clé étrangère - soit égale à une colonne (ou plusieurs) d'une autre table de la relation - la clé primaire à laquelle se réfère la clé étrangère. Cette clause peut présenter des conditions supplémentaires. Ce type de jointure est appelé équijointure. Par exemple, pour rechercher des informations contenues dans les tables Instructor et Department avec une seule instruction SQL, vous effectueriez une jointure entre ces deux tables.

La syntaxe générale de l'instruction SELECT permet de joindre plus de deux tables :

SELECT *liste_sélection*
FROM *table1*, *table2*, ..., *tableN*
WHERE *table1.colonne1* = *table2.colonne2* AND
...
table2.colonne3 = *tableN.colonneN*
...
conditions_additionnelles

Les variables sont définies ainsi :

- *liste_sélection* est l'ensemble des colonnes et des expressions contenues dans *table1* à *tableN*.
- *table1* à *tableN* représentent les tables à partir desquelles les valeurs de colonnes sont recherchées.
- *colonne1* à *colonneN* représentent les colonnes dans *table1* à *tableN* qui sont associées et qui remplissent la condition sur laquelle se fonde la recherche.
- *conditions_additionnelles* sont des critères de recherche optionnels.

Notez qu'il n'est pas nécessaire de spécifier les *colonne1* à *colonneN* dans *liste_sélection*.

NB :
Pour réaliser des opérations de jointure, on pourra également utiliser l'instruction JOIN comme cela est illustré dans les exemples qui suivent.

7.5.2 Jointure simple de deux tables

7.5.2.1 Etude de la syntaxe sur base d'exemples

Dans l'exemple qui suit, on joint des données provenant des tables « Class » et « Instructor ». Différentes syntaxes sont envisageables.

```
SELECT Class_ID, Course_ID, Last_Name, First_Name, Position
FROM Class, Instructor
WHERE Class.Instructor_ID = Instructor.Instructor_ID
ORDER BY Class_ID;
```

OU

Utilisation d'alias (technique du « "surnommage ») :

```
SELECT Class_ID, Course_ID, Last_Name, First_Name, Position
FROM Class C, Instructor I
WHERE C.Instructor_ID = I.Instructor_ID
ORDER BY Class_ID;
```

```
CLASS_ID            COURS LAST_NAME               FIRST_NAME              POSITION
------------------- ----- ----------------------- ----------------------- --------------------
-
103400              183   JASON                   JERROLD                 ASSOCIATE PROFESSOR
103600              50    CHANG                   ROGER                   ASSISTANT PROFESSOR
104200              198   RICHARDSON              NANCY                   ASSISTANT PROFESSOR
104500              184   RESTON                  RAPHAEL                 ASSOCIATE PROFESSOR
108300              101   CHERNOW                 BESS                    ASSOCIATE PROFESSOR
108400              115   TORRES                  PETER                   PROFESSOR
108600              183   JASON                   JERROLD                 ASSOCIATE PROFESSOR
109100              101   WEISS                   ROBERTA                 PROFESSOR
110300              199   CHU                     STEVEN                  PROFESSOR
120200              199   PARKER                  WILLIAM                 PROFESSOR
```

OU

Utilisation de JOIN

Jointure interne :

```
SELECT Class_ID, Course_ID, Last_Name, First_Name, Position
FROM Class C
INNER JOIN Instructor I
ON C.Instructor_ID = I.Instructor_ID;
```

OU

Jointure naturelle (n'existe que pour certains systèmes de gestion de bases de données) :

```
SELECT Class_ID, Course_ID, Last_Name, First_Name, Position
FROM Class
NATURAL JOIN Instructor;
```

Remarque

Quand on réalise une jointure entre deux tables, on n'est pas obligé de sélectionner des données issues de toutes les tables jointes.

```
SELECT distinct Last_Name, First_Name, Position
FROM Class C, Instructor I
WHERE C.Instructor_ID = I.Instructor_ID;
```

```
LAST_NAME       FIRST_NAME      POSITION
--------------- --------------- --------------------
JASON           JERROLD         ASSOCIATE PROFESSOR
CHANG           ROGER           ASSISTANT PROFESSOR
RICHARDSON      NANCY           ASSISTANT PROFESSOR
RESTON          RAPHAEL         ASSOCIATE PROFESSOR
CHERNOW         BESS            ASSOCIATE PROFESSOR
TORRES          PETER           PROFESSOR
JASON           JERROLD         ASSOCIATE PROFESSOR
WEISS           ROBERTA         PROFESSOR
CHU             STEVEN          PROFESSOR
PARKER          WILLIAM         PROFESSOR
```

OU

```
SELECT distinct Last_Name, First_Name, Position
FROM Class C
INNER JOIN Instructor I
ON C.Instructor_ID = I.Instructor_ID;
```

7.5.2.2 Colonnes ambiguës

Dans une jointure, aucune référence de colonne ne doit être ambiguë. Dans ce contexte, il y a ambiguïté lorsque l'on se réfère à une colonne existant dans plus d'une table de la jointure. Cette ambiguïté est levée en qualifiant la colonne avec le nom de la table concernée. Le système Oracle produit un message d'erreur lorsqu'une référence est ambiguë. Le listing figurant ci-après illustre cette situation pour la colonne Instructor_ID.

```
SELECT Instructor_ID, Position
FROM Class C, Instructor I
WHERE C.Instructor_ID = I.Instructor_ID
ORDER BY Class_ID;

Msg 209, Niveau 16, État 1, Ligne 1
Nom de colonne 'Instructor_ID' ambigu.
```

Syntaxe correcte :

```
SELECT I.Instructor_ID, Position
FROM Class C, Instructor I
WHERE C.Instructor_ID = I.Instructor_ID
ORDER BY Class_ID;
```

7.5.3 Produit cartésien de deux tables

Erreur classique quand on débute avec les jointures : oubli de la condition de jointure

```
SELECT Class_ID, Course_ID, Last_Name, First_Name, Position
FROM Class, Instructor
ORDER BY Class_ID;
```

Le nombre de lignes renvoyées peut être très important.

7.5.4 Requêtes sur des données récursives

Soit la table TRAVAILLEUR.

TRAVAILLEUR							
MATRICULE	NOM	PRENOM	FONCTION	SUPERIEUR	DATEENTREE	SALAIRE	DEPART
950	Batini	Paul	Employé	1650	7/11/81	1375	30
1000	Dupont	René	Directeur	1350	23/06/74	3500	50
1050	Dasnoy	Anne	Employée	1000	26/07/79	1750	50
1100	Legrand	Fernand	Ouvrier	1700	1/06/87	1375	20
1150	Bechet	Robert	Ouvrier	1700	3/07/75	1625	20
1200	Gandois	Paul	Ouvrier	1700	30/10/84	1500	20
1250	Bechot	François	Ouvrier	1700	29/11/89	1250	20

TRAVAILLEUR							
MATRICULE	NOM	PRENOM	FONCTION	SUPERIEUR	DATEENTREE	SALAIRE	DEPART
1300	Bechot	Grégoire	Ouvrier	1700	13/06/91	1200	20
1350	Durant	Paul	Président		19/03/71	6750	40
1400	Hilot	Corine	Secrétaire	1350	23/05/71	1875	40
1450	Dasnoy	Sophie	Secrétaire	1700	20/05/79	1250	20
1500	Legrand	Vanessa	Secrétaire	1000	19/03/94	1000	50
1550	Lepetit	Sylvie	Secrétaire	1000	29/11/91	1125	50
1600	Fosty	Albertine	Secrétaire	1650	19/04/90	1175	60
1650	Fontaine	Gabriel	Directeur	1350	3/07/73	3875	60
1700	Fensie	Maurice	Ingénieur	1350	17/07/89	2875	20
1750	Rossion	Jean	Ingénieur	1350	19/07/86	3125	10
1800	Pirsoul	Suzanne	Employée	1750	5/11/84	1375	10
1850	Cerf	Jacqueline	Employée	1000	6/08/80	1425	50
1900	Rouma	Fernando	Employé	1750	23/04/71	1500	10
1950	Clauwaert	Albert	Employé	1750	25/05/75	1475	10
2000	Volard	Frédéric	Employé	1650	17/03/73	1500	30
2050	Fluzy	Christophe	Employé	1650	20/06/94	1250	30

Objectif : afficher le nom et le prénom des différents travailleurs avec en vis-à-vis de chacun d'eux le nom et le prénom de leur supérieur hiérarchique.

```
SELECT sub.NOM, sub.PRENOM, sup.NOM, sup.PRENOM
FROM travailleur sub, travailleur sup
WHERE sub.superieur=sup.matricule;
```

OU

```
SELECT sub.NOM, sub.PRENOM, sup.NOM, sup.PRENOM
FROM travailleur sub
inner join travailleur sup
on sub.superieur=sup.matricule;

NOM        PRENOM     NOM        PRENOM
---------- ---------- ---------- ----------
Dupont     René       Dupont     René
Dasnoy     Anne       Dasnoy     Anne
Legrand    Fernand    Legrand    Fernand
Bechet     Robert     Bechet     Robert
Gandois    Paul       Gandois    Paul
Bechot     François   Bechot     François
Bechot     Grégoire   Bechot     Grégoire
Hilot      Corine     Hilot      Corine
Dasnoy     Sophie     Dasnoy     Sophie
Legrand    Vanessa    Legrand    Vanessa
Lepetit    Sylvie     Lepetit    Sylvie
Fosty      Albertine  Fosty      Albertine
Fontaine   Gabriel    Fontaine   Gabriel
...
```

Il faut constater que le Président de la société n'apparaît pas dans le résultat.

7.5.5 Jointure externe

Objectif :

Afficher le NOM et le PRENOM des différents clients avec le numéro des commandes qu'ils ont passées. **On veut que les clients qui n'ont jamais commandé soient affichés également.**

```
SELECT nom, prenom, ncom
FROM client cli
LEFT OUTER JOIN Commande com
ON cli.ncli=com.ncli
ORDER BY nom, prenom;
```

Notes

Suivant les moteurs, on peut également disposer de jointure croisée (CROSS JOIN / produit cartésien) et de jointure d'union (UNION JOIN / équivalent à FULL OUTER JOIN).

7.5.6 Comment interpréter une requête complexe ?

Ces requêtes étant relativement complexes on pourra, pour interpréter leur signification, considérer utilement la procédure d'évaluation suivante (il s'agit d'une évaluation fictive, le SGBD utilisant généralement d'autres procédés plus efficaces pour construire le résultat).

Pour une requête mono-table :

- on considère la table spécifiée dans la clause FROM;
- on sélectionne les lignes sur base de la clause WHERE;
- on classe ces lignes en groupes comme spécifié dans la clause GROUP BY;
- on ne retient que les groupes qui vérifient la clause HAVING;
- de chacun de ces groupes retenus, on extrait les valeurs demandées dans la clause SELECT;
- les valeurs demandées sont finalement éventuellement ordonnées.

Pour une requête multi-tables :

- on considère les tables spécifiées dans la clause FROM;
- on effectue les jointures en fonction de ce qui est défini dans les clauses WHERE ou au niveau de JOIN;
- on sélectionne les lignes de la jointure sur base des autres conditions de la clause WHERE;
- on classe ces lignes en groupes comme spécifié dans la clause GROUP BY;
- on ne retient que les groupes qui vérifient la clause HAVING;
- de chacun de ces groupes retenus, on extrait les valeurs demandées dans la clause SELECT;
- les valeurs demandées sont finalement éventuellement ordonnées.

Les notes ci-dessus présentent chaque fois une manière de décortiquer une requête. Cependant, il faut remarquer qu'une même requête peut être étudiée de différentes façons. De plus, il est important de rappeler que la manière avec laquelle on analyse une requête peut parfois être bien différente de la façon avec laquelle le moteur du SGBD va l'exécuter.

7.6 Instruction SELECT avec opérateur d'ensemble

Le langage SQL est une implémentation partielle du modèle relationnel comme l'avait imaginé Codd, le fondateur de la théorie relationnelle. La version SQL d'Oracle fournit trois opérateurs d'ensembles : INTERSECT, UNION, et MINUS.

7.6.1 Opérateur INTERSECT

instruction_select1
INTERSECT
instruction_select2
[*clause ORDER BY*]

Les variables sont définies ainsi :
- ✓ *instruction_select1* et *instruction_select2* représentent des instructions SELECT valides.
- ✓ *clause_order_by* est une clause ORDER BY qui se réfère aux colonnes par des numéros au lieu d'utiliser le nom des colonnes.

Voici quelques exigences et aspects liés à l'utilisation de l'opérateur INTERSECT :
- Les deux instructions SELECT ne peuvent pas contenir de clause ORDER BY. Toutefois, vous pouvez classer les résultats de l'opération entière.
- Le nombre de colonnes renvoyées par *instruction_select1* doit être égal au nombre de colonnes renvoyées par *instruction_select2*.
- Les types de données des colonnes de *instruction_select1* doivent correspondre à ceux des colonnes de *instruction_select2*.
- L'élément optionnel *clause ORDER BY* diffère de la clause ORDER BY habituelle d'une instruction SELECT, car les colonnes utilisées pour le classement doivent être référencées par numéro plutôt que par un nom. La raison est que SQL ne requiert pas que les noms des colonnes renvoyées par *instruction_select1* soient identiques à ceux de *instruction_select2*. Par conséquent, vous devez vous référer aux colonnes à utiliser dans le classement des résultats par leur position dans la liste de sélection.

Analysez le scénario suivant, qui illustre l'utilisation de l'opérateur INTERSECT. En tant que secrétaire de l'université Flugle, vous suspectez que certains enseignants travaillent aussi à l'université rivale Hoover. Une liste des enseignants de l'université Hoover tombe entre vos mains. Dans le listing suivant, la première requête renvoie les enseignants de l'université Flugle, et la seconde ceux de l'université Hoover. La troisième requête utilise l'opérateur INTERSECT pour rechercher les enseignants communs aux deux tables.

```
SELECT Last_Name, First_Name, MI, Position
FROM Instructor;

LAST_NAME       FIRST_NAME M POSITION
--------------- ---------- - -------------------------
HITCHCOCK       BORIS        PROFESSOR
DANIELS         LAURA        ASSOCIATE PROFESSOR
EDWARDS         SAMANTHA     PROFESSOR
CHANG           ROGER        ASSISTANT PROFESSOR
JASON           JERROLD      ASSOCIATE PROFESSOR
TORRES          PETER        PROFESSOR
ANGELO          ARTHUR       ASSOCIATE PROFESSOR
RICHARDSON      NANCY        ASSISTANT PROFESSOR
PARKER          WILLIAM      PROFESSOR
CHERNOW         BESS         ASSOCIATE PROFESSOR
CHU             STEVEN       PROFESSOR
WEISS           ROBERTA      PROFESSOR
RESTON          RAPHAEL      ASSOCIATE PROFESSOR
BILLINGS        BENJAMIN     ASSISTANT PROFESSOR
YOUNG           ALLAN        ASSOCIATE PROFESSOR
NILAND          MARTINA      ASSOCIATE PROFESSOR
BATES           JOSEPH       ASSISTANT PROFESSOR
POULSON         RANIER       PROFESSOR
```

```
SELECT Last_Name, First_Name, MI, Position
FROM Hoover_Instructor;

LAST_NAME                 FIRST_NAME       M POSITION
------------------------- ---------------- - --------------------
CHANG                     ROGER              ASSISTANT PROFESSOR
JASON                     JERROLD            ASSOCIATE PROFESSOR
TORRES                    PETER              PROFESSOR
HERNOW                    BESS               ASSOCIATE PROFESSOR
CHU                       STEVEN             PROFESSOR
RESTON                    RAPHAEL            ASSOCIATE PROFESSOR
YOUNG                     ALLAN              ASSOCIATE PROFESSOR
NILAND                    MARTINA            ASSOCIATE PROFESSOR
BATES                     JOSEPH             ASSISTANT PROFESSOR
HUSTON                    MYRON             T PROFESSOR
RABINOWITZ                KERMIT            A LECTURER
CHRISTIANSON              PAUL              V PROFESSOR

SELECT Last_Name, First_Name, MI, Position
FROM Instructor
INTERSECT
SELECT Last_Name, First_Name, MI, Position
FROM Hoover_Instructor;

LAST_NAME                 FIRST_NAME       M POSITION
------------------------- ---------------- - --------------------
BATES                     JOSEPH             ASSISTANT PROFESSOR
CHANG                     ROGER              ASSISTANT PROFESSOR
CHERNOW                   BESS               ASSOCIATE PROFESSOR
CHU                       STEVEN             PROFESSOR
JASON                     JERROLD            ASSOCIATE PROFESSOR
NILAND                    MARTINA            ASSOCIATE PROFESSOR
RESTON                    RAPHAEL            ASSOCIATE PROFESSOR
TORRES                    PETER              PROFESSOR
YOUNG                     ALLAN              ASSOCIATE PROFESSOR
```

7.6.2 Opérateur UNION

Tôt ou tard, vous pouvez vous trouver dans une situation où vous aurez besoin de combiner les lignes de tables de structures semblables pour produire un rapport ou pour créer une table à des fins d'analyse. Même si ces tables contiennent des informations similaires, leur teneur peut différer considérablement. Afin d'accomplir cette tâche, vous pouvez recourir à l'opérateur UNION.

La syntaxe de cet opérateur est assez simple :

instruction_select1
UNION
instruction_select2
[*clause ORDER BY*]

Les variables sont définies ainsi :

- *instruction_select1* et *instruction_select2* représentent des instructions SELECT valides.
- *clause ORDER BY* est une clause ORDER BY qui se réfère aux colonnes par un numéro plutôt que par un nom.

Cet opérateur combine les lignes renvoyées par la première instruction SELECT avec celles de la seconde. Gardez à l'esprit les éléments suivants lorsque vous utilisez cet opérateur:

- Les deux instructions SELECT ne peuvent pas contenir de clause ORDER BY. Toutefois, vous pouvez classer les résultats de l'opération entière.
- Le nombre de colonnes renvoyées par *instruction-select1* doit être égal au nombre de colonnes renvoyées par *instruction_select2*.
- Les types de données des colonnes de *instruction-select1* doivent correspondre à ceux des colonnes de *instruction_select2*.
- L'élément optionnel *clause ORDER BY* diffère de la clause ORDER BY habituelle d'une instruction SELECT, car les colonnes utilisées pour le classement doivent être référencées par un numéro plutôt que par un nom. La raison est que SQL ne requiert pas que les noms des colonnes renvoyées par *instruction-select1* soient identiques à ceux de *instruction_select2*.

7.6.3 Opérateur EXCEPT

Cet opérateur permet aussi d'effectuer des comparaisons entre deux ensembles de lignes.

La syntaxe de cet opérateur ressemble à celle des opérateurs INTERSECT et UNION :

instruction_select1
EXCEPT
instruction_select2
[*clause ORDER BY*]

Les variables sont définies ainsi :

- *instruction_select1* et *instruction-select2* représentent des instructions SELECT valides.
- *clause ORDER BY* est une clause ORDER BY qui se réfère aux colonnes par un numéro plutôt que par un nom.

Les exigences et caractéristiques d'utilisation de cet opérateur sont essentiellement les mêmes que celles vues précédemment pour les opérateurs INTERSECT et UNION. Pour illustrer le propos, reprenons les tables d'enseignants des universités Flugle et Hoover, respectivement Instructor et Hoover_Instructor. Vous souhaitez rechercher les enseignants de Flugle ne travaillant pas à Hoover. Le listing suivant montre comment exploiter l'opérateur MINUS à cette fin. La requête renvoie 9 lignes.

```
SELECT Last_Name, First_Name, MI, Position
FROM Instructor
EXCEPT
SELECT Last_Name, First_Name, MI, Position
FROM Hoover_Instructor;

LAST_NAME       FIRST_NAME M POSITION
--------------- ---------- - ------------------------
ANGELO          ARTHUR       ASSOCIATE PROFESSOR
BILLINGS        BENJAMIN     ASSISTANT PROFESSOR
DANIELS         LAURA        ASSOCIATE PROFESSOR
EDWARDS         SAMANTHA     PROFESSOR
HITCHCOCK       BORIS        PROFESSOR
PARKER          WILLIAM      PROFESSOR
POULSON         RANIER       PROFESSOR
RICHARDSON      NANCY        ASSISTANT PROFESSOR
WEISS           ROBERTA      PROFESSOR
```

Toutefois, la requête du listing précédent n'indique rien à propos des enseignants de Hoover qui ne travaillent pas à Flugle. Pour obtenir ces informations, les deux instructions SELECT doivent être inversées (voir ci-après).

```
SELECT Last_Name, First_Name, MI, Position
FROM Hoover_Instructor
EXCEPT
SELECT Last_Name, First_Name, MI, Position
FROM Instructor;

LAST_NAME       FIRST_NAME M POSITION
--------------- ---------- - ---------
CHRISTIANSON    PAUL       V PROFESSOR
HUSTON          MYRON      T PROFESSOR
RABINOWITZ      KERMIT     A LECTURER
```

Cet exemple démontre que l'opérateur EXCEPT ne vous indique pas toutes les différences entre les deux tables. Il ne fournit que les lignes du premier ensemble qui ne peuvent pas être trouvées dans le second. Vous pouvez aussi spécifier une clause WHERE dans les instructions SELECT utilisées avec l'opérateur EXCEPT.

7.7 Création d'une table à partir d'une autre table

Comme cela é déjà été vu, pour créer une table à partir d'une autre, il suffit d'utiliser SELECT ... INTO

Exemple :

```
SELECT * INTO client_archive FROM client
```

Note.
Avec Oracle par exemple, la syntaxe est différente. Il faut utiliser CREATE TABLE basé sur une sous-requête.

7.8 Résumé

Ce chapitre a été consacré aux caractéristiques suivantes du langage SQL :

- Le système fournit des fonctions de groupe telles que MAX et MIN renvoyant une seule valeur pour un ensemble global de lignes.
- L'instruction SELECT dispose de deux clauses optionnelles GROUP BY et HAVING qui fournissent des lignes groupées par des valeurs communes de colonnes données.
- Les données hiérarchiques peuvent être organisées par l'intermédiaire de la clause CONNECT BY. Vous devez indiquer une condition de départ avec la clause START WITH. Utilisable avec Oracle pas avec SQL Server.
- Une opération de jointure représente une instruction SELECT dont la clause FROM spécifie deux ou plusieurs tables, associées à une ou plusieurs conditions de jointure qui se basent généralement sur une égalité des valeurs de clés primaires et étrangères.
- L'opérateur INTERSECT retourne l'ensemble de lignes communes à deux ensembles de lignes.
- L'opérateur UNION retourne les lignes distinctes qui existent dans deux ensembles de lignes.
- L'opérateur EXCEPT renvoie les lignes d'un ensemble qui ne figurent pas dans un autre ensemble.

III Mise en pratique

1 Présentation

Pour rappel, lors de la modélisation des données, trois niveaux sont abordés :

- niveau conceptuel ;
- niveau logique ;
- niveau physique.

Le **modèle conceptuel** des données est un modèle qui prend en compte les règles de gestion assignées au système d'information mais **pas les contraintes technologiques** de réalisation d'une base de données. **Au niveau conceptuel, le choix d'un type de stockage n'est donc pas encore réalisé et encore moins la sélection d'un outil particulier (SQL Server, MySQL, PostgreSQL...).**

Le **modèle logique** des données correspond au choix d'un type de base de données. Dans les cas qui nous concernent, il s'agira de bases de données relationnelles. Concrètement, **au niveau logique, nous définissons les tables à créer**. Dans ce cas, « modèle relationnel » est synonyme de « modèle logique ».

L'étape de passage au **modèle physique** des données consiste en la prise en compte des contraintes liées au matériel de traitement et aux outils logiciels choisis. Concrètement, les types de données sont définis, la longueur des champs, la création d'index est envisagée, le nom de certaines entités, certains attributs peut être adapté...

2 Contexte

Dans les pages qui suivent, nous envisageons la gestion de données concernant des clients, des produits et des commandes.

3 Création du modèle conceptuel des données (MCD)

3.1 Recueil des informations

Toute analyse débute par de nombreuses interviews et par la collecte de documents, de fichiers...

D'autre part, il faut expliciter de façon claire **les règles de gestion**.

Exemples :

Règle de gestion 1 : Un client peut passer une ou plusieurs commandes ou aucune commande. On envisage par exemple le cas de clients potentiels qui auraient pu demander une offre de prix mais n'ont encore rien commandé.

Règle de gestion 2 : Une commande peut inclure un ou plusieurs produits.

Règle de gestion 3 : Une commande est passée par un et un seul client.

Règle de gestion 4 : Sur un bon de commande, ne peut figurer qu'une seule ligne pour un produit donné.

3.2 Constitution du dictionnaire des données

Afin de constituer le dictionnaire des données, il faut récolter tous les documents, tous les fichiers... qui circulent au sein de l'entreprise. Ces derniers sont porteurs des données que le système d'information devra permettre de gérer.

Exemple :

NOM	SIGNIFICATION	TYPE	LONGUEUR	NATURE	RÈGLE DE CALCUL OU INTÉGRITÉ
NCLI	Identifiant du client	AN	4	E	
TITRE	Titre du client (Madame, Monsieur)	A	10	E	
NOM	Nom du client	A	50	E	

NOM	SIGNIFICATION	TYPE	LONGUEUR	NATURE	RÈGLE DE CALCUL OU INTÉGRITÉ
PRENOM	Prénom du client	A	50	E	
RUE	Rue	AN	120	E	
CP	Code postal	AN	7	E	
LOCALITE	Localité	AN	50	E	
CAT	Catégorie	AN	2	E	
COMPTE	Solde du compte client	N	(8,2)		
NPRO	Identifiant du produit	AN	5	E	
LIBELLE	Désignation du produit	AN	90	E	
PRIX	Prix « catalogue » d'un produit	N	(7,2)	E	
QSTOCK	Quantité en stock d'un produit	N	Valeur entière inférieure à 10000		
NCOM	Numéro de commande	N	Valeur entière inférieure à 10^6	E	
DATE	Date de la commande	D		E	
QCOM	Quantité commandée	N	Valeur entière inférieure à 10000	E	
PRIX_PRO_COM	Prix du produit commandé	N	(7,2)	E	
MONTANT	Montant pour le produit concerné	N		CA	PRIX_PRO_COM X QCOM
TOTAL	Total pour la commande	N		CA	Somme des montants

Signification de certaines abréviations :

Type :
A : alphabétique
AN : alphanumérique
D : date
N : numérique

Nature :
E : élémentaire
CO :concaténé
CA : calculé

3.3 Graphe des dépendances fonctionnelles

On dit qu'un attribut Y dépend fonctionnellement d'un attribut (ou d'un groupe d'attributs) X si à une valeur de X correspond une et une seule valeur de Y.

Exemple :

L'attribut « nom du client » est en dépendance fonctionnelle par rapport au « n° client ». En effet, à un numéro de client, il n'y a qu'un seul nom de client associé.

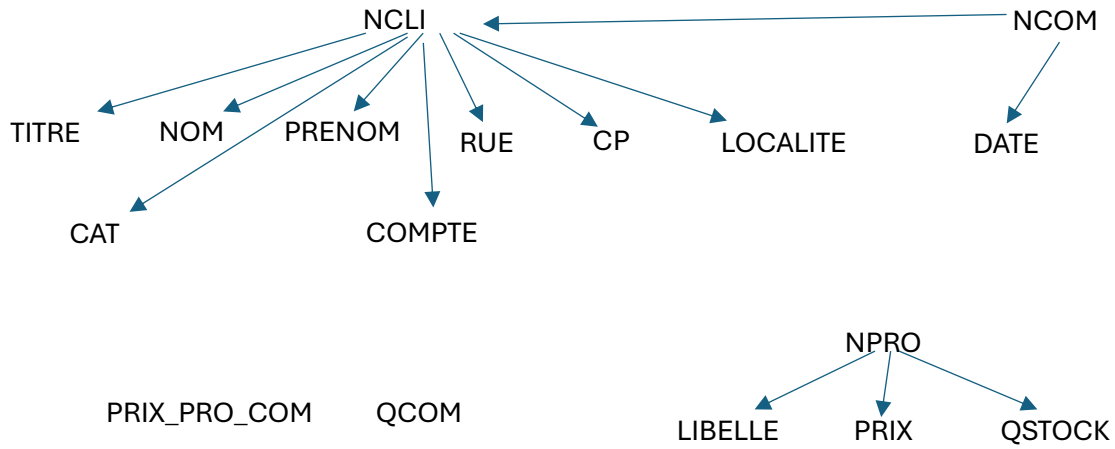
Autre exemple :

L'attribut « quantité commandée » est en dépendance fonctionnelle par rapport au groupe d'attributs « n° commande » et « n° de produit ».

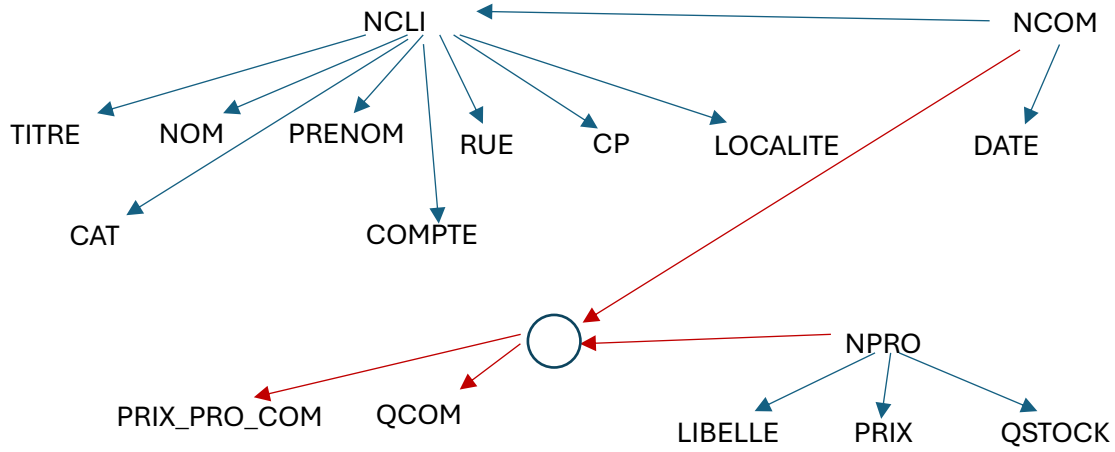
En pratique :

On extrait du dictionnaire des données les propriétés qui ne sont ni calculées ni concaténées.
On établit ensuite la liste des dépendances fonctionnelles dont le domaine de départ ne contient qu'une seule propriété (pas de combinaison de propriétés).

On obtient :

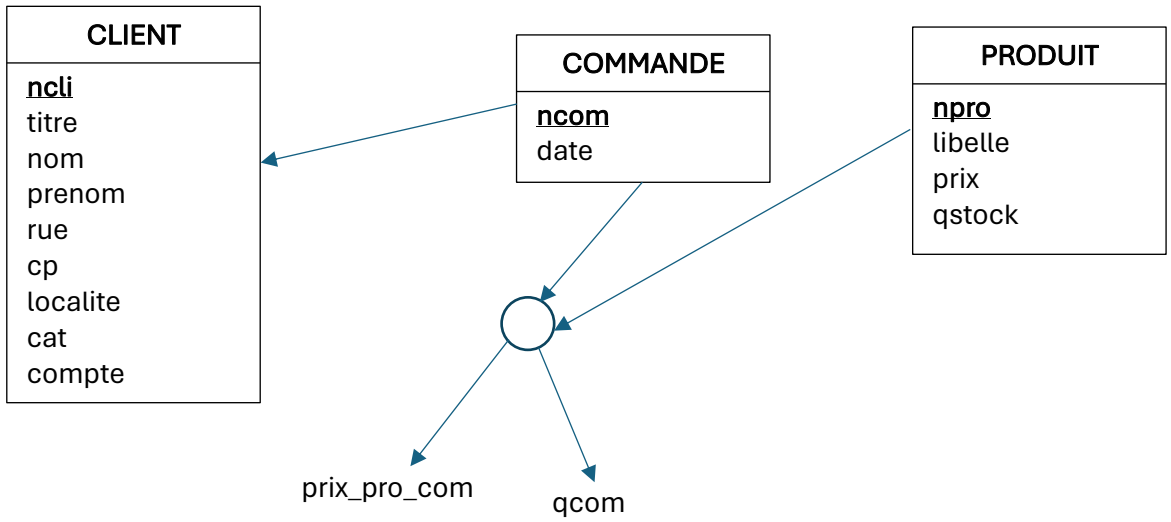


S'il reste des propriétés isolées, on recherche des dépendances fonctionnelles qui conduisent à ces propriétés **à partir de propriétés combinées**. Si on n'en trouve pas pour une propriété, celle-ci reste isolée.
NB : Il faut toujours éviter les cycles au niveau des SAT (structures d'accès théoriques).



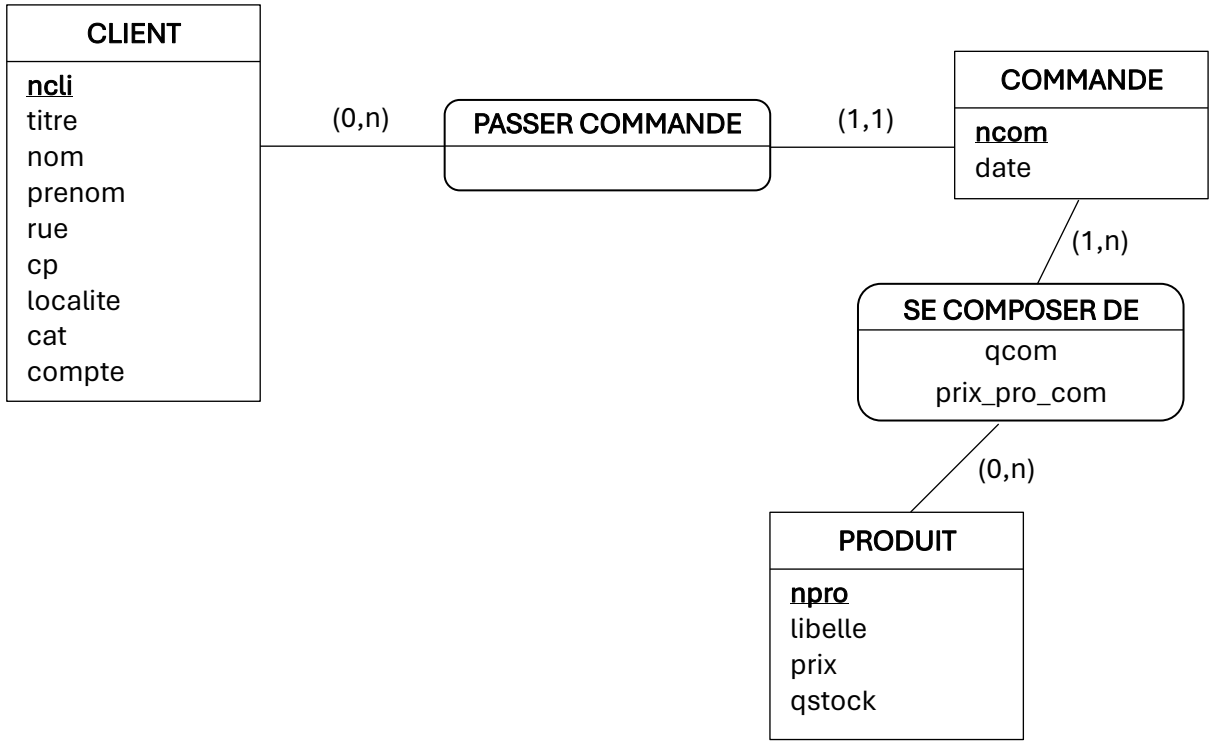
3.4 Construction du modèle conceptuel des données (MCD)

Partant du graphe des dépendances fonctionnelles, on commence par représenter les entités (individus) en spécifiant pour chacun d'eux les propriétés adéquates.



Les arcs restants mettent en évidence les **relations**. Les propriétés non isolées restantes sont affectées à des relations. Les propriétés isolées constituent quant à elles des entités isolées.

Les règles de gestion doivent permettre de trouver les cardinalités.



4 Création du modèle logique des données (MLD)

4.1 Présentation, méthodologie et exemple

Le modèle logique des données a pour but de représenter l'univers des données décrit dans le MCD mais en tenant compte du type de base de données choisi : relationnel, objet...

Dans le cas présent, à ce stade du niveau logique, nous irons vers une base de données de type relationnelle.

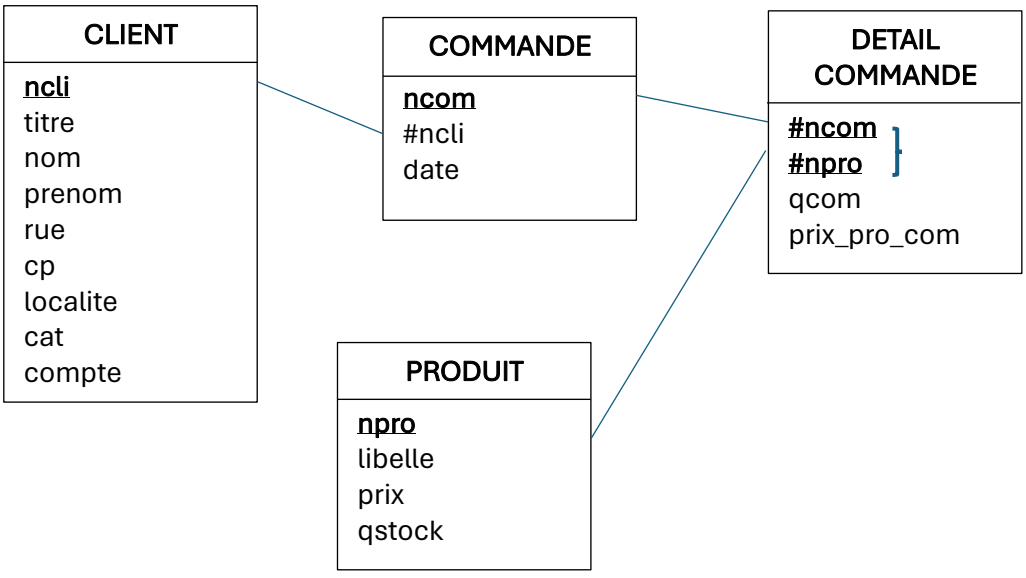
Document intéressant concernant le passage du MCD vers le MLD : https://www-igm.univ-mlv.fr/~chochois/RessourcesCommunes/BDD/Modelisation/coursMLD.pdf

Voici un résumé de quelques points.

- Au niveau des entités :
  - Une entité (individu) du MCD devient une table.
  - L'identifiant de l'entité devient la clé primaire de la table.
  - Les autres propriétés de l'entité deviennent les attributs (les champs) de la table.

- Au niveau des associations (relations) :
  - Une association de type 1 à plusieurs (cardinalités maximales à « 1 » d'un côté de l'association et à « n » de l'autre côté) se traduit par la création d'une clé étrangère dans la table qui correspond à l'entité côté « 1 ». **Dans l'exemple qui nous concerne, nous avons ce phénomène au niveau de l'association (relation) qui existe entre les entités COMMANDE et CLIENT. Une clé étrangère sera créée dans la table qui contient les commandes.**
  - Une association de type plusieurs à plusieurs (cardinalités maximales à « n » des deux côtés de l'association) se traduit par la création d'une table dont la clé primaire est composée des clés étrangères concernées dans les entités mises en relation. Les éventuelles propriétés de l'association deviennent des attributs (champs) de cette table. **Dans l'exemple qui nous concerne, nous avons ce phénomène au niveau de l'association (relation) qui existe entre les entités COMMANDE et PRODUIT. Une table reprenant le détail des commandes sera créée.**

En appliquant ces règles, voici les tables que nous obtiendrons.



4.2 Les formes normales

4.2.1 Principe et utilité de la normalisation

- La normalisation consiste à faire éclater une table en plusieurs tables plus réduites et cela en utilisant les formes normales et **à condition bien évidemment que cela se justifie**.
- L'objectif de la normalisation est de limiter le nombre de redondances d'informations afin d'optimiser notamment les opérations de mises à jour des données.
- Les principes de la normalisation et de l'algèbre relationnelle ont été mis au point par Messieurs Codd et Date.
- Il peut arriver que l'on décide de dénormaliser quand par exemple il faut privilégier la vitesse de consultation des données (au détriment des opérations de mise à jour).

4.2.2 Les formes normales

En général, on peut distinguer cinq ou six formes normales. Ci-après sont présentées les trois premières.

4.2.2.1 La première forme normale (1 FN)

Une table relationnelle est dite de première forme normale si chacun de ses attributs dépend fonctionnellement de la clé ET est non répétitif pour une occurrence de la clé.

Exemple :

La table suivante n'est pas en 1 FN :

PÈRE
n° national
nom des enfants

L'attribut « nom des enfants » est répétitif, un père peut en effet avoir plusieurs enfants => pas en 1 FN.

4.2.2.2 La deuxième forme normale (2 FN)

Une table relationnelle est dite en 2° forme normale si chacun de ses attributs non-clé dépend complètement de la clé (multi-composant).

Exemple :

PRODUIT
n° produit
n° fournisseur
prix du produit
nom du fournisseur

- La clé est constituée de l'association des attributs « n° produit » et « n° fournisseur ».
- Il ne s'agit pas d'une table de 2 FN, en effet l'attribut « nom du fournisseur » ne dépend pas complètement de la clé, il ne dépend que d'une partie de la clé (n° fournisseur).

4.2.2.3 La troisième forme normale (3 FN)

Une table relationnelle est dite en 3° forme normale si elle est en 2° FN **ET** s'il n'y a pas de dépendance fonctionnelle entre ses attributs non-clé.

Exemple :

LIVRE
n° identification
titre
code de l'auteur
nom de l'auteur

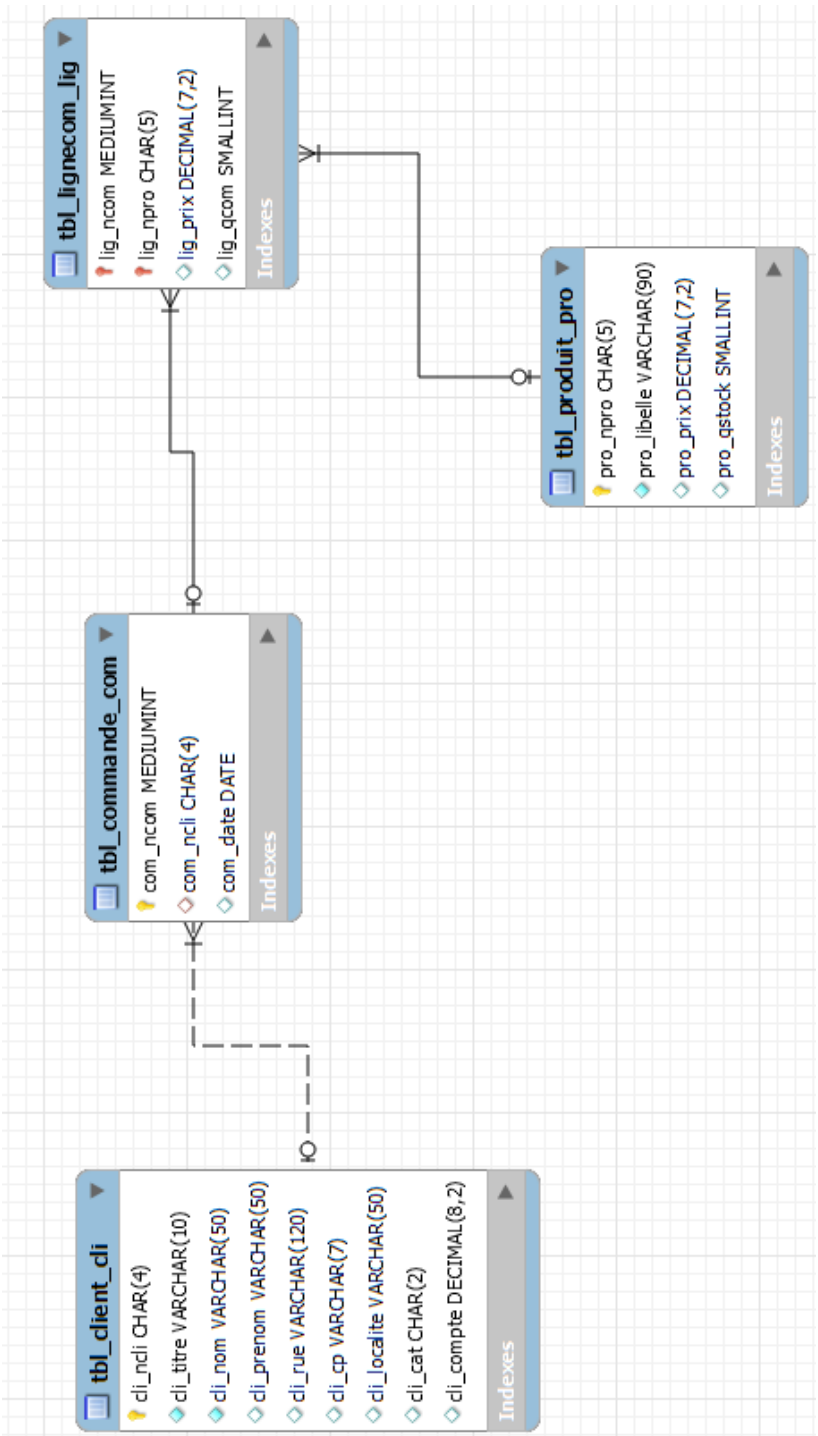
Il y a une dépendance fonctionnelle entre le « code de l'auteur » et le « nom de l'auteur » => pas en 3 FN.

5 Création du modèle physique des données

Au niveau physique, nous allons au-delà du choix d'un type de base de données (effectué au niveau logique). Pour rappel, dans le cas présent, il s'agira d'une base de données relationnelle. Nous pouvons ajouter que le système de gestion de base de données utilisé sera SQL Server. **À ce niveau, nous tiendrons donc compte des contraintes liées à l'utilisation de cet outil (types de champs proposés, performances de ce SGBD...).**

Voici un aperçu des tables qui seront créées via les scripts présentés dans la suite.

Nous pouvons constater que la dénomination des entités, des champs... peut être modifiée afin de faciliter la gestion ultérieure de la base de données mais également en pensant à faciliter la création et la maintenance du code créé au sein des applications qui accèderont à cette base de données.

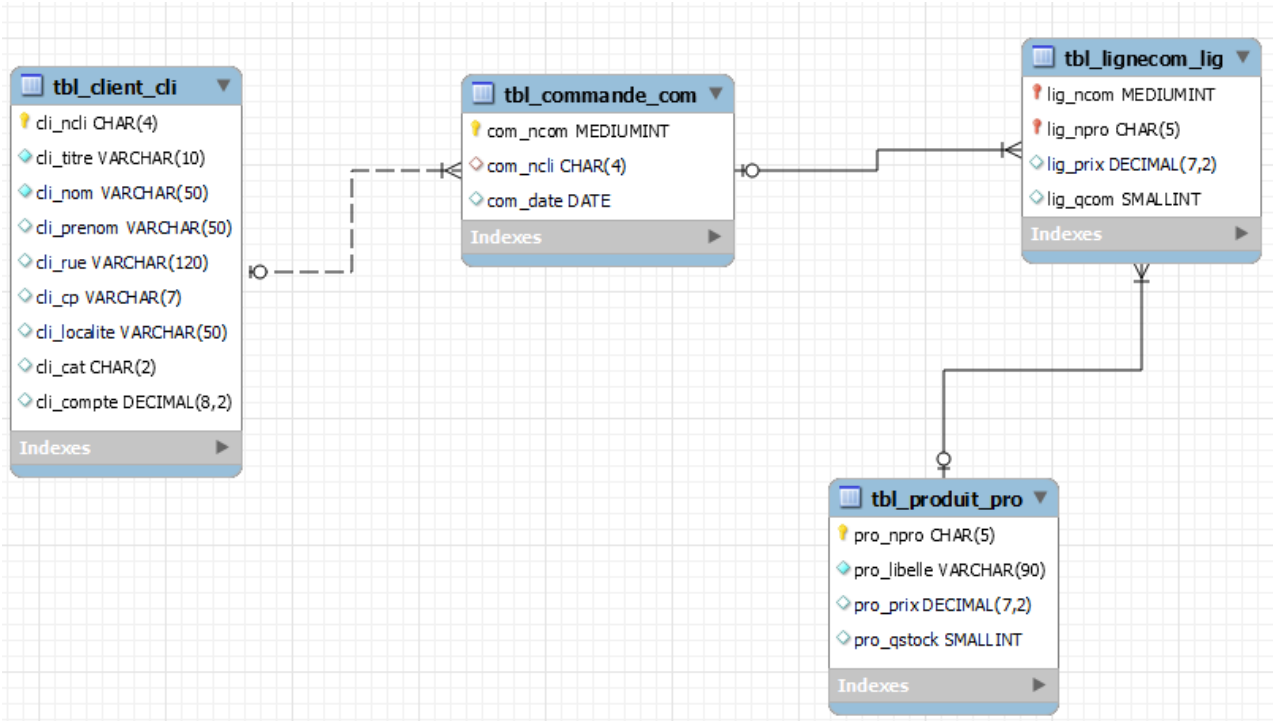


6 Implémentation de la base de données

6.1 Présentation de la base de données créée – Niveau physique de la modélisation des données

Voici un aperçu de la base de données « bd_commande ».

Structure :



Contenu des tables :

Table « tbl_client_cli »

cli_ncli	cli_titre	cli_nom	cli_prenom	cli_rue	cli_cp	cli_localite	cli_cat	cli_compte
B062	Madame	Goffin	Jacqueline	Rue de la Gare 72	5000	Namur	B2	-3 200,00
B112	Monsieur	Hansenne	Albert	Rue A. Dumont 23	5300	Andenne	C1	1 250,00
B332	Monsieur	Monti	Paul	Rue Neuve 13	4970	Stavelot	B2	0,00
B512	Monsieur	Gillet	Grégoire	Rue de la Gare 14	4000	Liège	B1	-8 700,00
C003	Monsieur	Avron	François	Chemin de la Cure 8	4000	Liège	B1	-1 700,00
C123	Monsieur	Mercier	Jean	Rue Lemaître 25	5000	Namur	C1	-2 300,00
C400	Madame	Ferard	Anne	Rue du Tertre 65	5300	Andenne	B2	350,00
D063	Madame	Nadoy	Jacqueline	Boulevard de la Poste 1	4000	Liège	C3	-2 250,00
F010	Madame	Toussaint	Corine	Rue Godefroid 5	5300	Andenne	C1	0,00
F011	Monsieur	Poncelet	Fernando	Clos des Erables 17	4000	Liège	B2	0,00
F400	Monsieur	Jacob	Daniel	Chemin du Moulin 78	3000	Louvain	C2	0,00
K111	Madame	Vanbist	Sylvie	Rue Florimont 81	4540	Amay	B1	720,00
K729	Monsieur	Neuman	Herbert	Rue Bransart 40	4000	Liège	C3	0,00
L422	Madame	Franck	Albertine	Rue de Wépion 60	5000	Namur	C1	0,00
S127	Monsieur	Vanderka	René	Avenue des Roses 3	5000	Namur	C1	-4 580,00
S712	Monsieur	Guillaume	Albert	Rue du Lac 14a	5100	Wépion	B1	0,00

Table « tbl_commande_com »

com_ncom	com_ncli	com_date
178	K111	2025-06-12
179	C400	2025-06-12
182	S127	2025-06-14
184	C400	2025-06-27
185	F011	2025-07-02
186	C400	2025-07-02
188	B512	2025-07-03

Table « tbl_produit_pro »

pro_npro	pro_libelle	pro_prix	pro_qstock
CS262	Chevron en sapin 300 x 4 x 6 cm	5,50	45
CS264	Chevron en sapin 300 x 6 x 6 cm	6,80	2 690
CS464	Chevron en sapin 400 x 4 x 6 cm	6,00	450
PA45	Pointes en acier 45 (1 kg)	4,00	580
PA60	Pointes en acier 60 (1 kg)	5,00	134
PH222	Planche en hêtre 200 x 2 x 6 cm	12,00	782
PS222	Planche en hêtre 200 x 3 x 6 cm	15,00	1 220

Table « tbl_lignecom_lig »

lig_ncom	lig_npro	lig_prix	lig_qcom
178	CS464	6,00	25
179	CS262	5,50	60
179	PA60	5,00	20
182	PA60	5,00	30
184	CS464	6,00	120
184	PA45	4,00	20
185	CS464	6,00	260
185	PA60	5,00	15
185	PS222	15,00	600
186	PA45	4,00	3
188	CS464	6,00	180
188	PA45	4,00	22
188	PA60	5,00	70
188	PH222	12,00	92

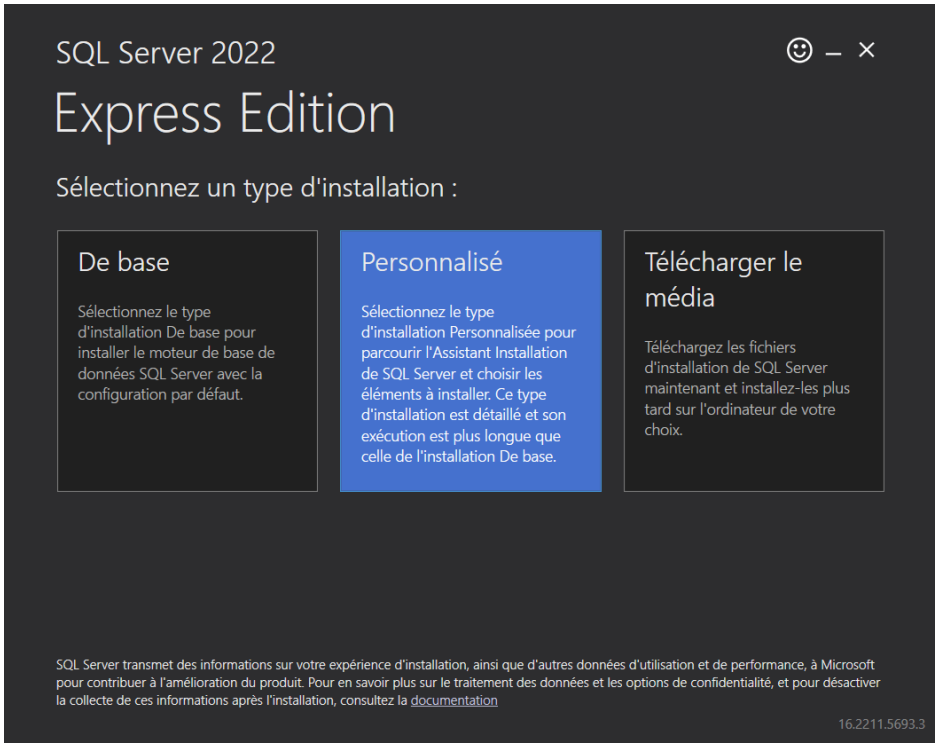
6.2 Outils utilisés

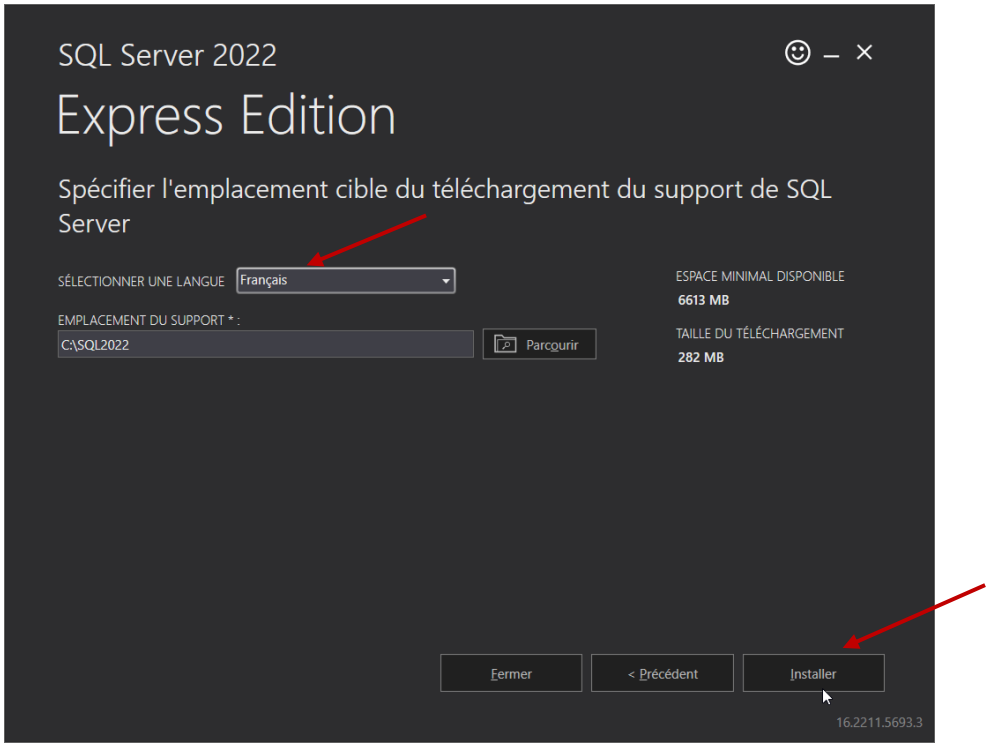
6.2.1 Installation du moteur du système de gestion de base(s) de données

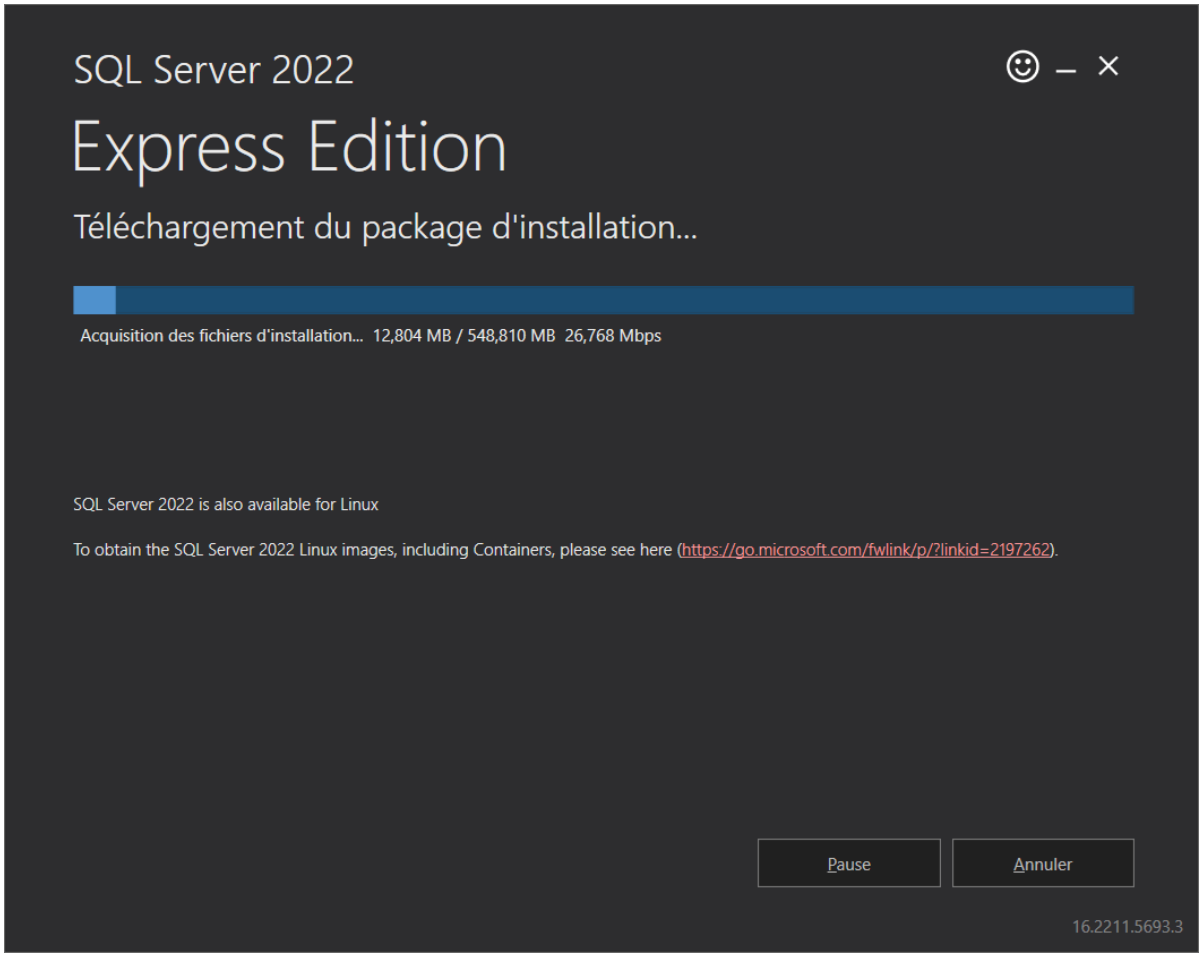
Dans le cas présent, c'est le système de gestion de données SQL Server 2022 Express Edition qui est utilisé. Il s'agit de la dernière version au moment de l'écriture de ces notes.
Lien : https://www.microsoft.com/fr-fr/sql-server/sql-server-downloads

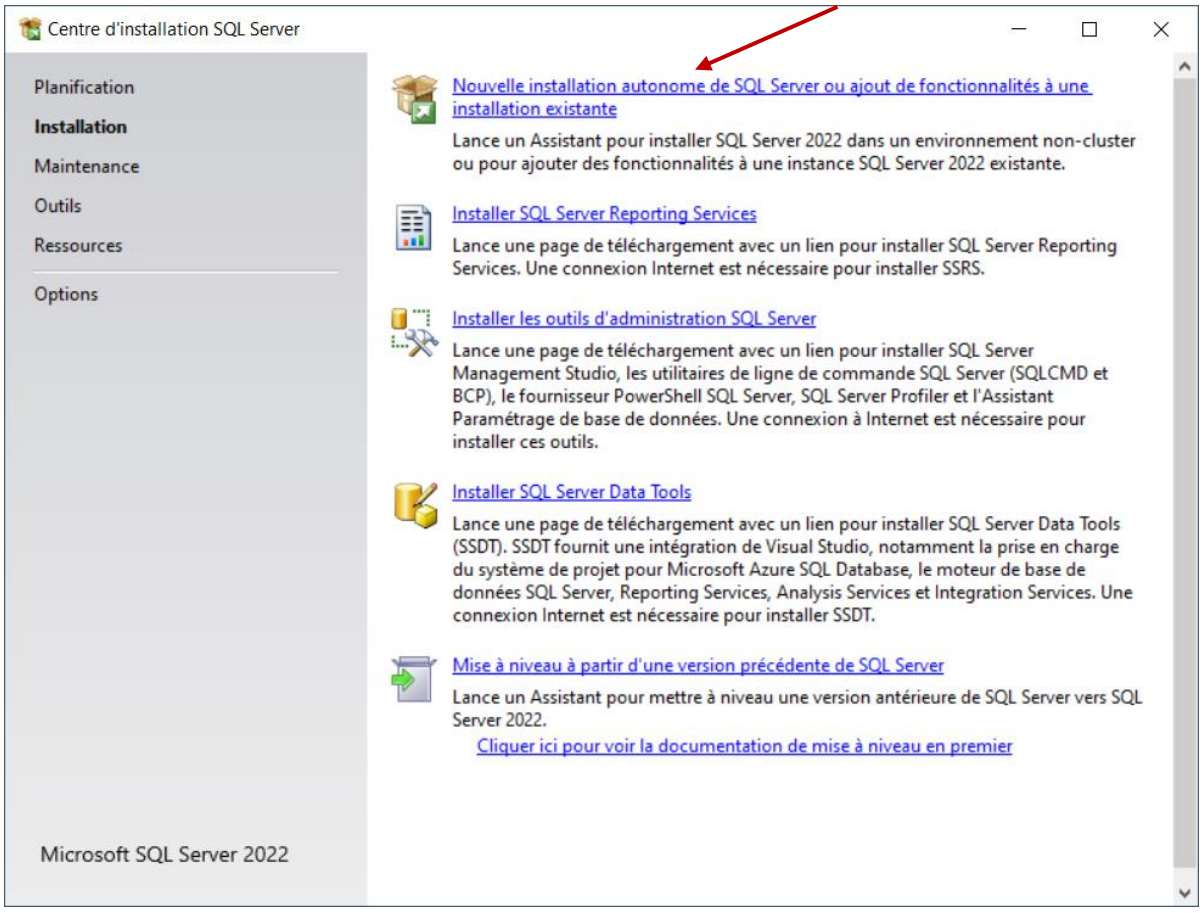
Voici un aperçu des différentes fenêtres rencontrées lors de l'installation.

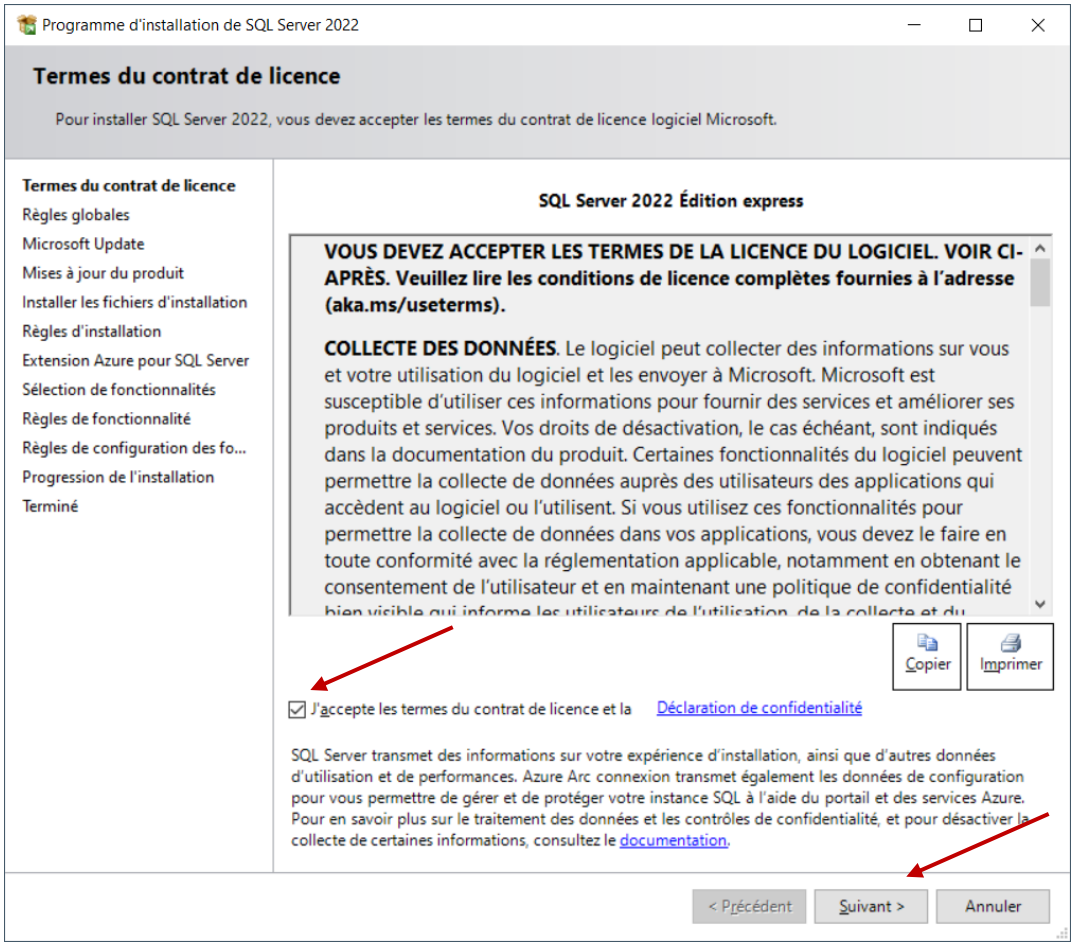
NB : Dans certains cas (le système affichera un avertissement), il peut être nécessaire de définir « France » et « Français (France) » au niveau des paramètres régionaux, de redémarrer la machine puis d'effectuer l'installation.

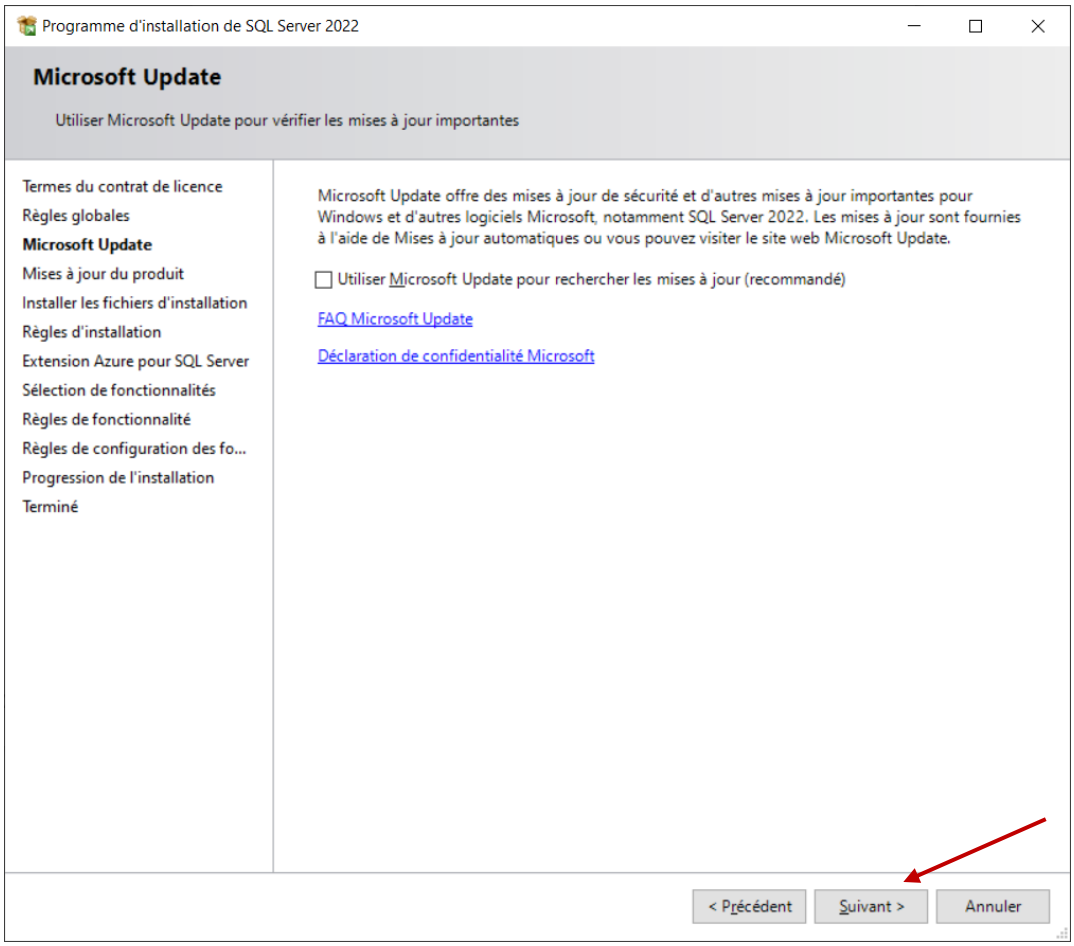


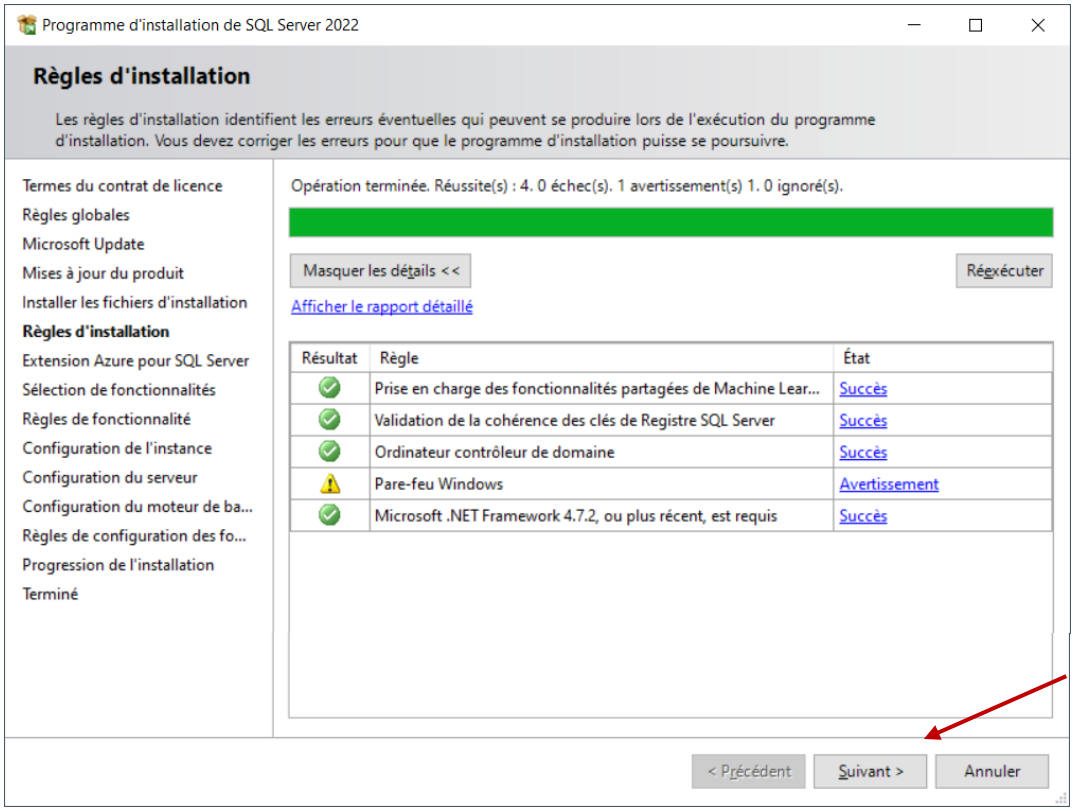


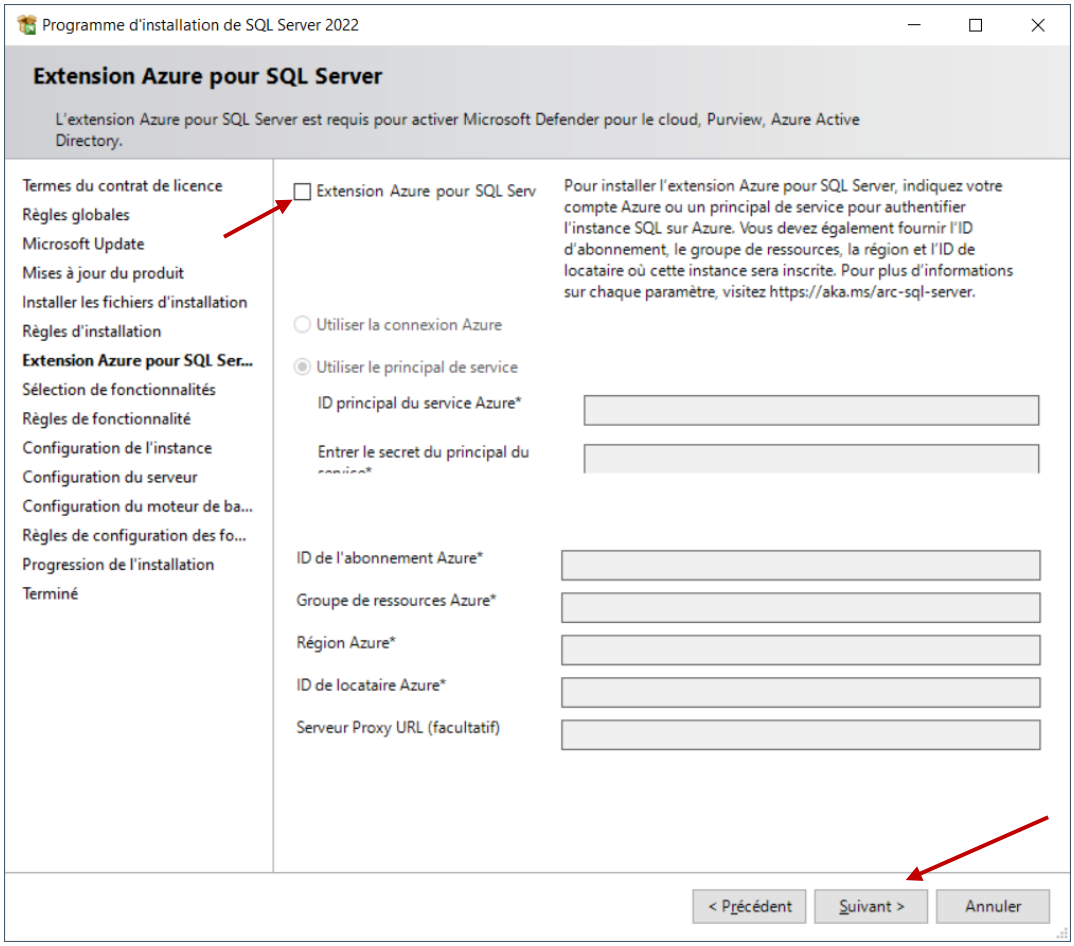

SQL Server 2022
Express Edition
Téléchargement du package d'installation...
Acquisition des fichiers d'installation... 12,804 MB / 548,810 MB 26,768 Mbps
SQL Server 2022 is also available for Linux
To obtain the SQL Server 2022 Linux images, including Containers, please see here (https://go.microsoft.com/fwlink/p/?linkid=2197262).
Pause
Annuler
16.2211.5693.3


Centre d'installation SQL Server
Planification
Installation
Maintenance
Outils
Ressources
Options
Microsoft SQL Server 2022
Nouvelle installation autonome de SQL Server ou ajout de fonctionnalités à une installation existante
Lance un Assistant pour installer SQL Server 2022 dans un environnement non-cluster ou pour ajouter des fonctionnalités à une instance SQL Server 2022 existante.
Installer SQL Server Reporting Services
Lance une page de téléchargement avec un lien pour installer SQL Server Reporting Services. Une connexion Internet est nécessaire pour installer SSRS.
Installer les outils d'administration SQL Server
Lance une page de téléchargement avec un lien pour installer SQL Server Management Studio, les utilitaires de ligne de commande SQL Server (SQLCMD et BCP), le fournisseur PowerShell SQL Server, SQL Server Profiler et l'Assistant Paramétrage de base de données. Une connexion à Internet est nécessaire pour installer ces outils.
Installer SQL Server Data Tools
Lance une page de téléchargement avec un lien pour installer SQL Server Data Tools (SSDT). SSDT fournit une intégration de Visual Studio, notamment la prise en charge du système de projet pour Microsoft Azure SQL Database, le moteur de base de données SQL Server, Reporting Services, Analysis Services et Integration Services. Une connexion Internet est nécessaire pour installer SSDT.
Mise à niveau à partir d'une version précédente de SQL Server
Lance un Assistant pour mettre à niveau une version antérieure de SQL Server vers SQL Server 2022.
Cliquer ici pour voir la documentation de mise à niveau en premier

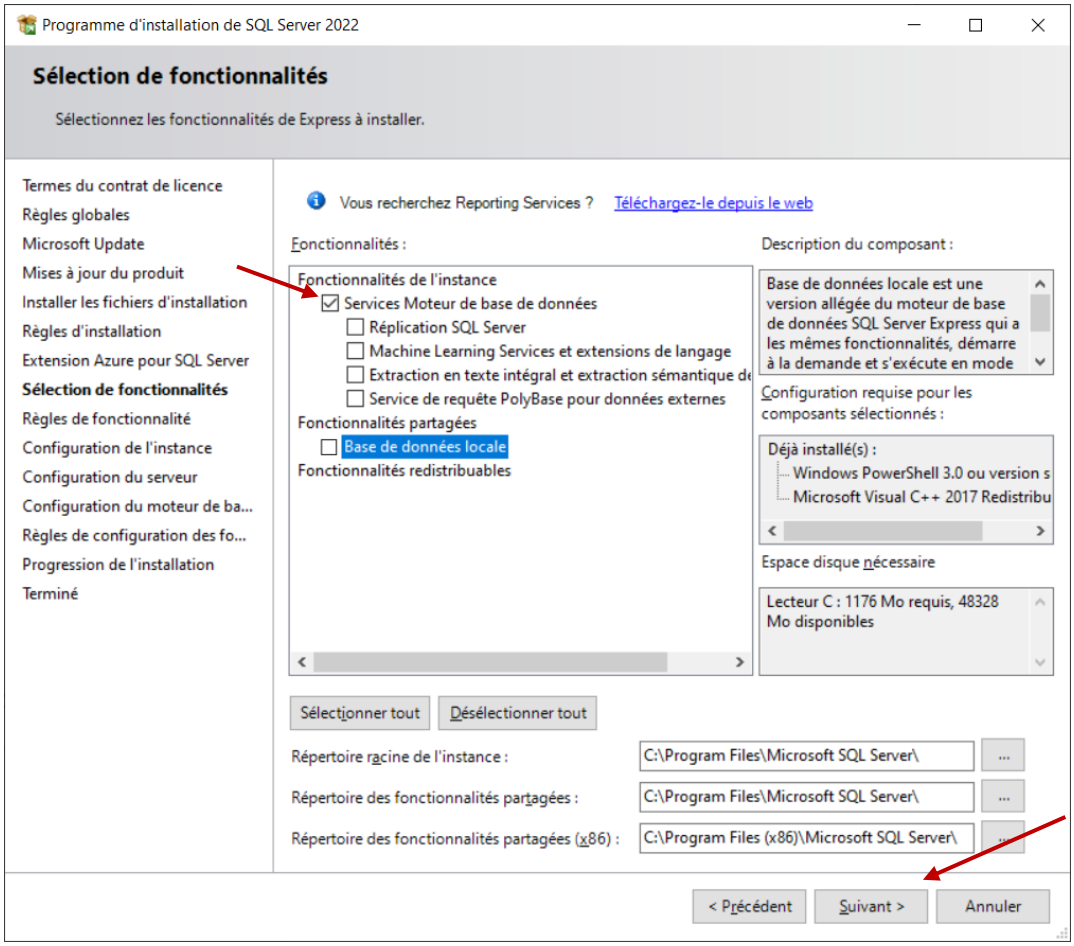

Programme d'installation de SQL Server 2022
Termes du contrat de licence
Pour installer SQL Server 2022, vous devez accepter les termes du contrat de licence logiciel Microsoft.
Termes du contrat de licence
Règles globales
Microsoft Update
Mises à jour du produit
Installer les fichiers d'installation
Règles d'installation
Extension Azure pour SQL Server
Sélection de fonctionnalités
Règles de fonctionnalité
Règles de configuration des fo...
Progression de l'installation
Terminé
SQL Server 2022 Édition express
VOUS DEVEZ ACCEPTER LES TERMES DE LA LICENCE DU LOGICIEL. VOIR CI-APRÈS. Veuillez lire les conditions de licence complètes fournies à l'adresse (aka.ms/useterms).
COLLECTE DES DONNÉES. Le logiciel peut collecter des informations sur vous et votre utilisation du logiciel et les envoyer à Microsoft. Microsoft est susceptible d'utiliser ces informations pour fournir des services et améliorer ses produits et services. Vos droits de désactivation, le cas échéant, sont indiqués dans la documentation du produit. Certaines fonctionnalités du logiciel peuvent permettre la collecte de données auprès des utilisateurs des applications qui accèdent au logiciel ou l'utilisent. Si vous utilisez ces fonctionnalités pour permettre la collecte de données dans vos applications, vous devez le faire en toute conformité avec la réglementation applicable, notamment en obtenant le consentement de l'utilisateur et en maintenant une politique de confidentialité
Copier
Imprimer
J'accepte les termes du contrat de licence et la
Déclaration de confidentialité
SQL Server transmet des informations sur votre expérience d'installation, ainsi que d'autres données d'utilisation et de performances. Azure Arc connexion transmet également les données de configuration pour vous permettre de gérer et de protéger votre instance SQL à l'aide du portail et des services Azure. Pour en savoir plus sur le traitement des données et les contrôles de confidentialité, et pour désactiver la collecte de certaines informations, consultez le documentation.
< Précédent
Suivant >
Annuler

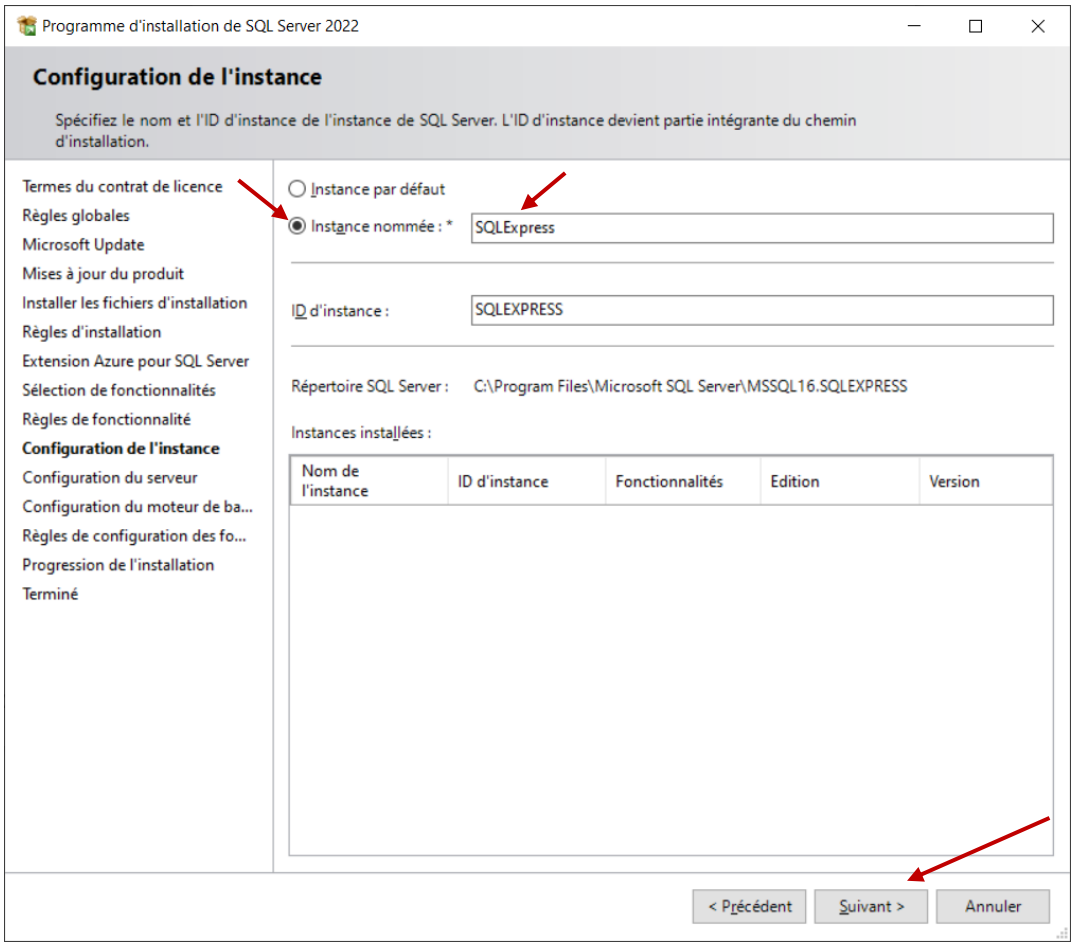

Programme d'installation de SQL Server 2022
Microsoft Update
Utiliser Microsoft Update pour vérifier les mises à jour importantes
Termes du contrat de licence
Règles globales
Microsoft Update
Mises à jour du produit
Installer les fichiers d'installation
Règles d'installation
Extension Azure pour SQL Server
Sélection de fonctionnalités
Règles de fonctionnalité
Règles de configuration des fo...
Progression de l'installation
Terminé
Microsoft Update offre des mises à jour de sécurité et d'autres mises à jour importantes pour Windows et d'autres logiciels Microsoft, notamment SQL Server 2022. Les mises à jour sont fournies à l'aide de Mises à jour automatiques ou vous pouvez visiter le site web Microsoft Update.
Utiliser Microsoft Update pour rechercher les mises à jour (recommandé)
FAQ Microsoft Update
Déclaration de confidentialité Microsoft
< Précédent
Suivant >
Annuler

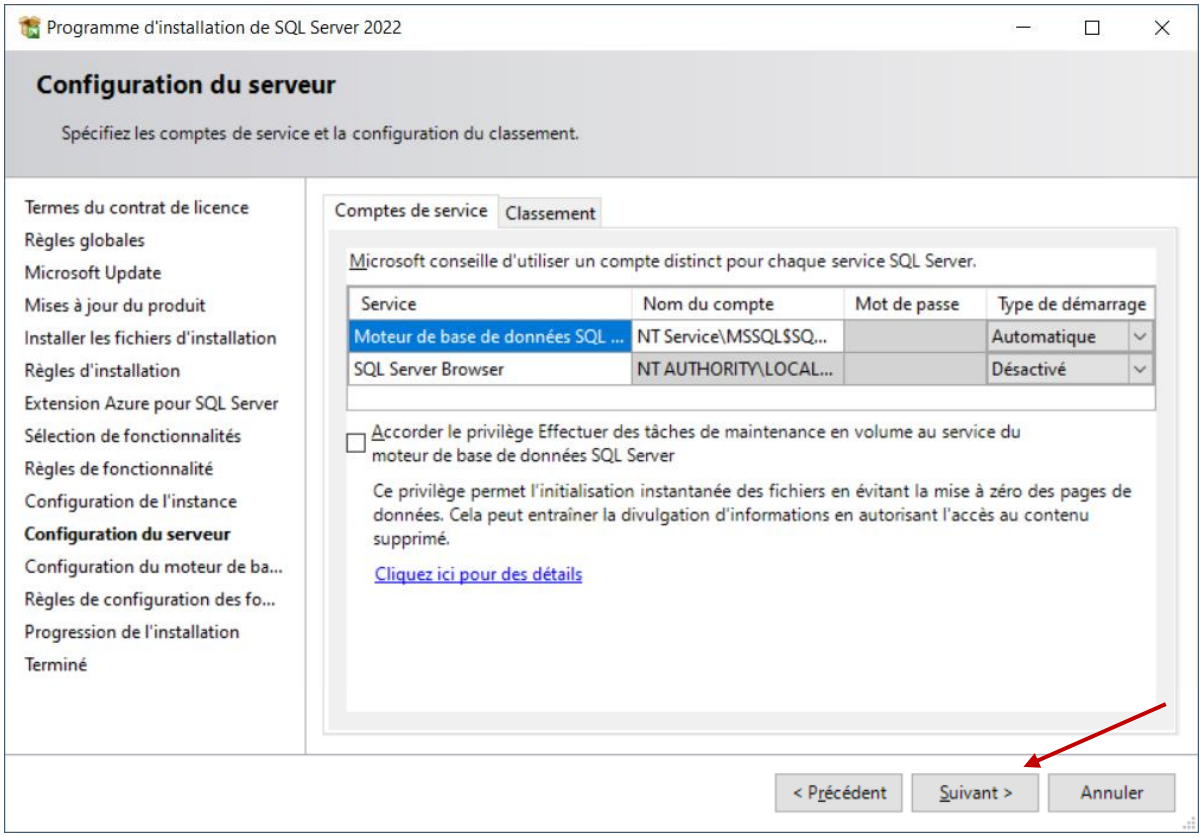


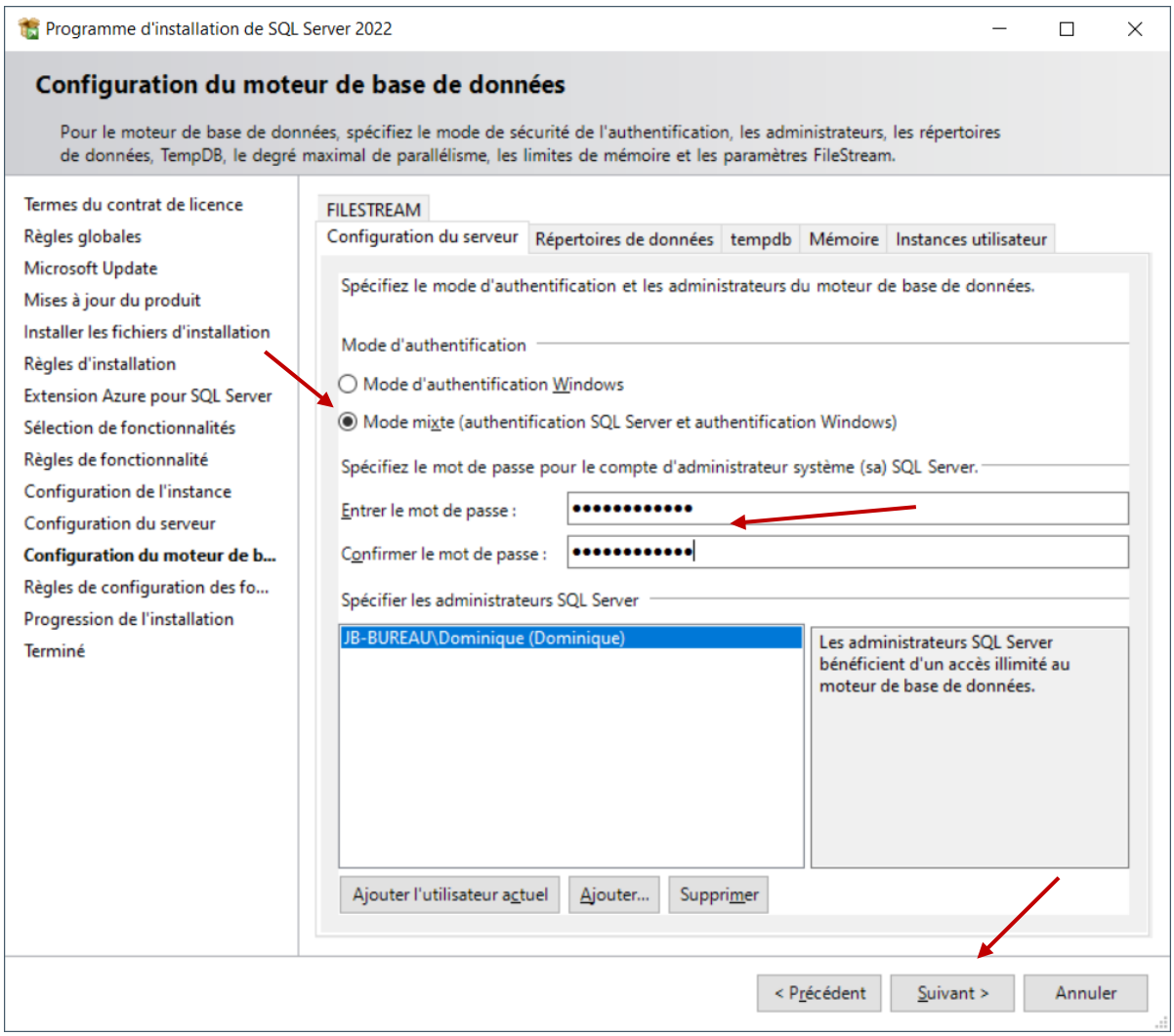


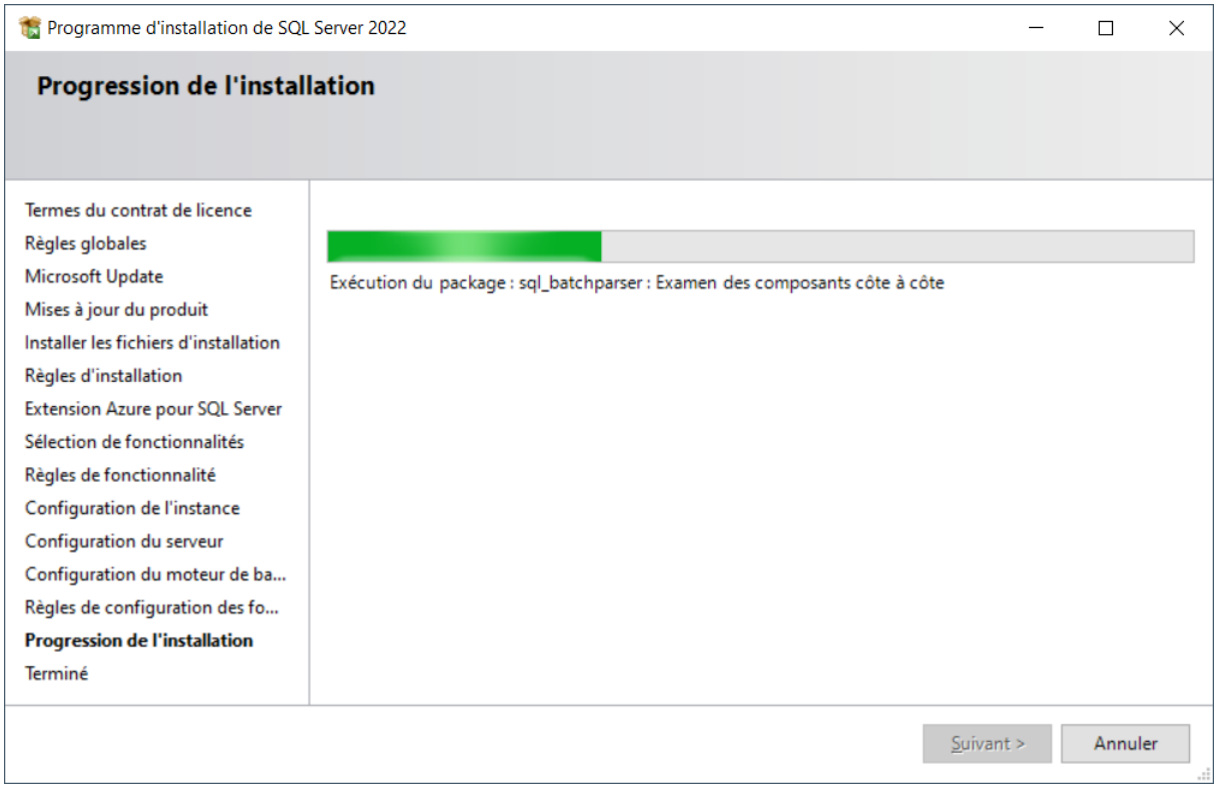
NB : Un compte Azure permet de se connecter à Microsoft Azure pour créer, gérer et superviser des ressources cloud telles que des machines virtuelles, bases de données, serveurs SQL, services d'intelligence artificielle, etc.

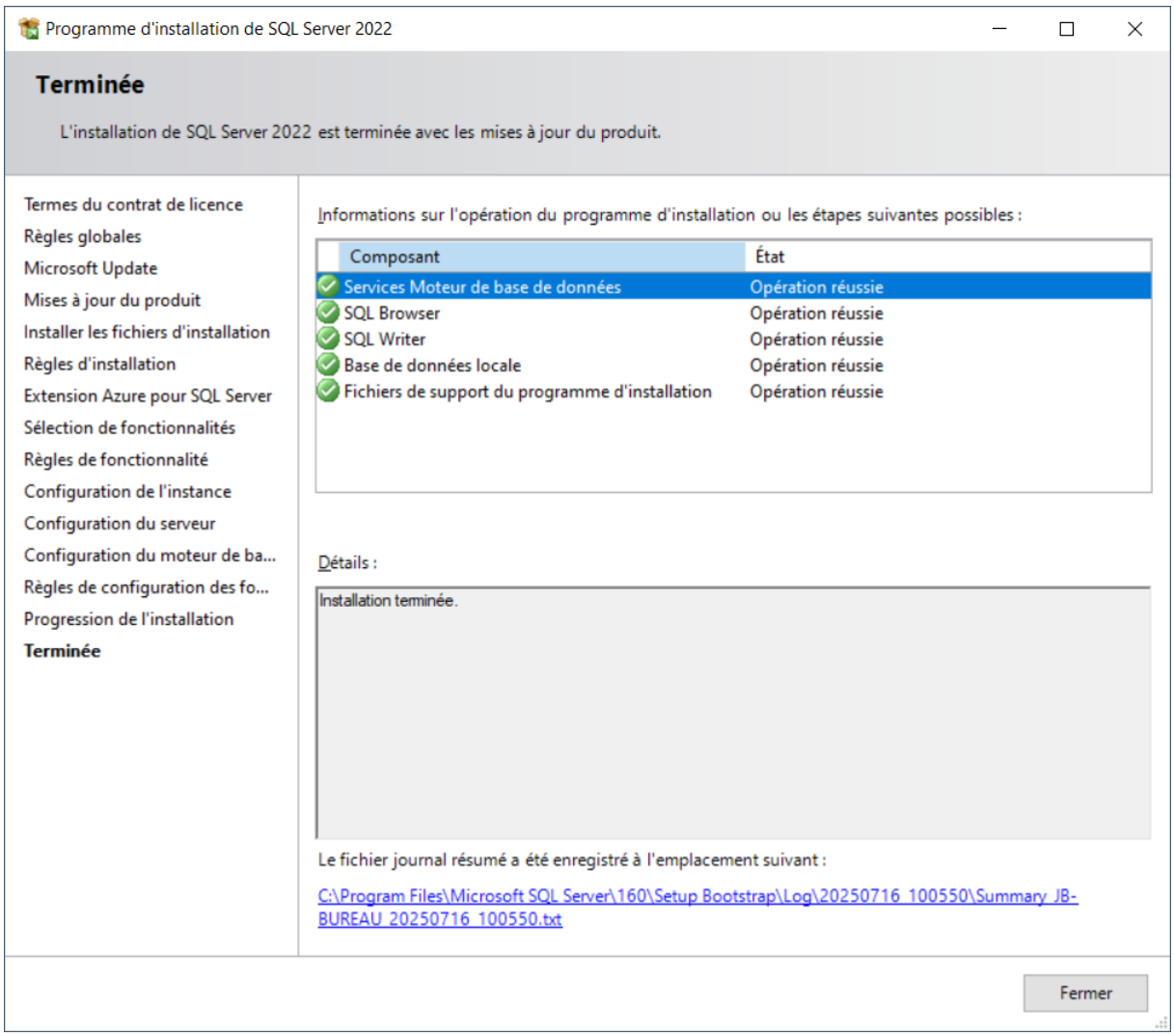

Programme d'installation de SQL Server 2022
Sélection de fonctionnalités
Sélectionnez les fonctionnalités de Express à installer.
Termes du contrat de licence
Règles globales
Microsoft Update
Mises à jour du produit
Installer les fichiers d'installation
Règles d'installation
Extension Azure pour SQL Server
Sélection de fonctionnalités
Règles de fonctionnalité
Configuration de l'instance
Configuration du serveur
Configuration du moteur de ba...
Règles de configuration des fo...
Progression de l'installation
Terminé
Vous recherchez Reporting Services ?
Téléchargez-le depuis le web
Fonctionnalités :
Fonctionnalités de l'instance
Services Moteur de base de données
Réplication SQL Server
Machine Learning Services et extensions de langage
Extraction en texte intégral et extraction sémantique de
Service de requête PolyBase pour données externes
Fonctionnalités partagées
Base de données locale
Fonctionnalités redistribuables
Description du composant :
Base de données locale est une version allégée du moteur de base de données SQL Server Express qui a les mêmes fonctionnalités, démarre à la demande et s'exécute en mode
Configuration requise pour les composants sélectionnés :
Déjà installé(s) :
Windows PowerShell 3.0 ou version s
Microsoft Visual C++ 2017 Redistribu
Espace disque nécessaire
Lecteur C : 1176 Mo requis, 48328 Mo disponibles
Sélectionner tout
Désélectionner tout
Répertoire racine de l'instance :
C:\Program Files\Microsoft SQL Server\
Répertoire des fonctionnalités partagées :
C:\Program Files\Microsoft SQL Server\
Répertoire des fonctionnalités partagées (x86) :
C:\Program Files (x86)\Microsoft SQL Server\
< Précédent
Suivant >
Annuler


Programme d'installation de SQL Server 2022
Configuration de l'instance
Spécifiez le nom et l'ID d'instance de l'instance de SQL Server. L'ID d'instance devient partie intégrante du chemin d'installation.
Termes du contrat de licence
Règles globales
Microsoft Update
Mises à jour du produit
Installer les fichiers d'installation
Règles d'installation
Extension Azure pour SQL Server
Sélection de fonctionnalités
Règles de fonctionnalité
Configuration de l'instance
Configuration du serveur
Configuration du moteur de ba...
Règles de configuration des fo...
Progression de l'installation
Terminé
Instance par défaut
Instance nommée : *
SQLExpress
ID d'instance :
SQLEXPRESS
Répertoire SQL Server :
C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS
Instances installées :
Nom de l'instance
ID d'instance
Fonctionnalités
Edition
Version
< Précédent
Suivant >
Annuler


Programme d'installation de SQL Server 2022
Configuration du serveur
Spécifiez les comptes de service et la configuration du classement.
Termes du contrat de licence
Règles globales
Microsoft Update
Mises à jour du produit
Installer les fichiers d'installation
Règles d'installation
Extension Azure pour SQL Server
Sélection de fonctionnalités
Règles de fonctionnalité
Configuration de l'instance
Configuration du serveur
Configuration du moteur de ba...
Règles de configuration des fo...
Progression de l'installation
Terminé
Comptes de service
Classement
Microsoft conseille d'utiliser un compte distinct pour chaque service SQL Server.
Service
Nom du compte
Mot de passe
Type de démarrage
Moteur de base de données SQL ...
NT Service\MSSQL$SQ...
Automatique
SQL Server Browser
NT AUTHORITY\LOCAL...
Désactivé
Accorder le privilège Effectuer des tâches de maintenance en volume au service du moteur de base de données SQL Server
Ce privilège permet l'initialisation instantanée des fichiers en évitant la mise à zéro des pages de données. Cela peut entraîner la divulgation d'informations en autorisant l'accès au contenu supprimé.
Cliquez ici pour des détails
< Précédent
Suivant >
Annuler


Programme d'installation de SQL Server 2022
Configuration du moteur de base de données
Pour le moteur de base de données, spécifiez le mode de sécurité de l'authentification, les administrateurs, les répertoires de données, TempDB, le degré maximal de parallélisme, les limites de mémoire et les paramètres FileStream.
Termes du contrat de licence
Règles globales
Microsoft Update
Mises à jour du produit
Installer les fichiers d'installation
Règles d'installation
Extension Azure pour SQL Server
Sélection de fonctionnalités
Règles de fonctionnalité
Configuration de l'instance
Configuration du serveur
Configuration du moteur de b...
Règles de configuration des fo...
Progression de l'installation
Terminé
FILESTREAM
Configuration du serveur
Répertoires de données
tempdb
Mémoire
Instances utilisateur
Spécifiez le mode d'authentification et les administrateurs du moteur de base de données.
Mode d'authentification
Mode d'authentification Windows
Mode mixte (authentification SQL Server et authentification Windows)
Spécifiez le mot de passe pour le compte d'administrateur système (sa) SQL Server.
Entrer le mot de passe :
Confirmer le mot de passe :
Spécifier les administrateurs SQL Server
JB-BUREAU\Dominique (Dominique)
Les administrateurs SQL Server bénéficient d'un accès illimité au moteur de base de données.
Ajouter l'utilisateur actuel
Ajouter...
Supprimer
< Précédent
Suivant >
Annuler


Programme d'installation de SQL Server 2022
Progression de l'installation
Termes du contrat de licence
Règles globales
Microsoft Update
Mises à jour du produit
Installer les fichiers d'installation
Règles d'installation
Extension Azure pour SQL Server
Sélection de fonctionnalités
Règles de fonctionnalité
Configuration de l'instance
Configuration du serveur
Configuration du moteur de ba...
Règles de configuration des fo...
Progression de l'installation
Terminé
Exécution du package : sql_batchparser : Examen des composants côte à côte
Suivant >
Annuler


Programme d'installation de SQL Server 2022
Terminée
L'installation de SQL Server 2022 est terminée avec les mises à jour du produit.
Termes du contrat de licence
Règles globales
Microsoft Update
Mises à jour du produit
Installer les fichiers d'installation
Règles d'installation
Extension Azure pour SQL Server
Sélection de fonctionnalités
Règles de fonctionnalité
Configuration de l'instance
Configuration du serveur
Configuration du moteur de ba...
Règles de configuration des fo...
Progression de l'installation
Terminée
Informations sur l'opération du programme d'installation ou les étapes suivantes possibles :
Composant
État
Services Moteur de base de données
Opération réussie
SQL Browser
Opération réussie
SQL Writer
Opération réussie
Base de données locale
Opération réussie
Fichiers de support du programme d'installation
Opération réussie
Détails :
Installation terminée.
Le fichier journal résumé a été enregistré à l'emplacement suivant :
C:\Program Files\Microsoft SQL Server\160\Setup Bootstrap\Log\20250716_100550\Summary_JB-BUREAU_20250716_100550.txt
Fermer

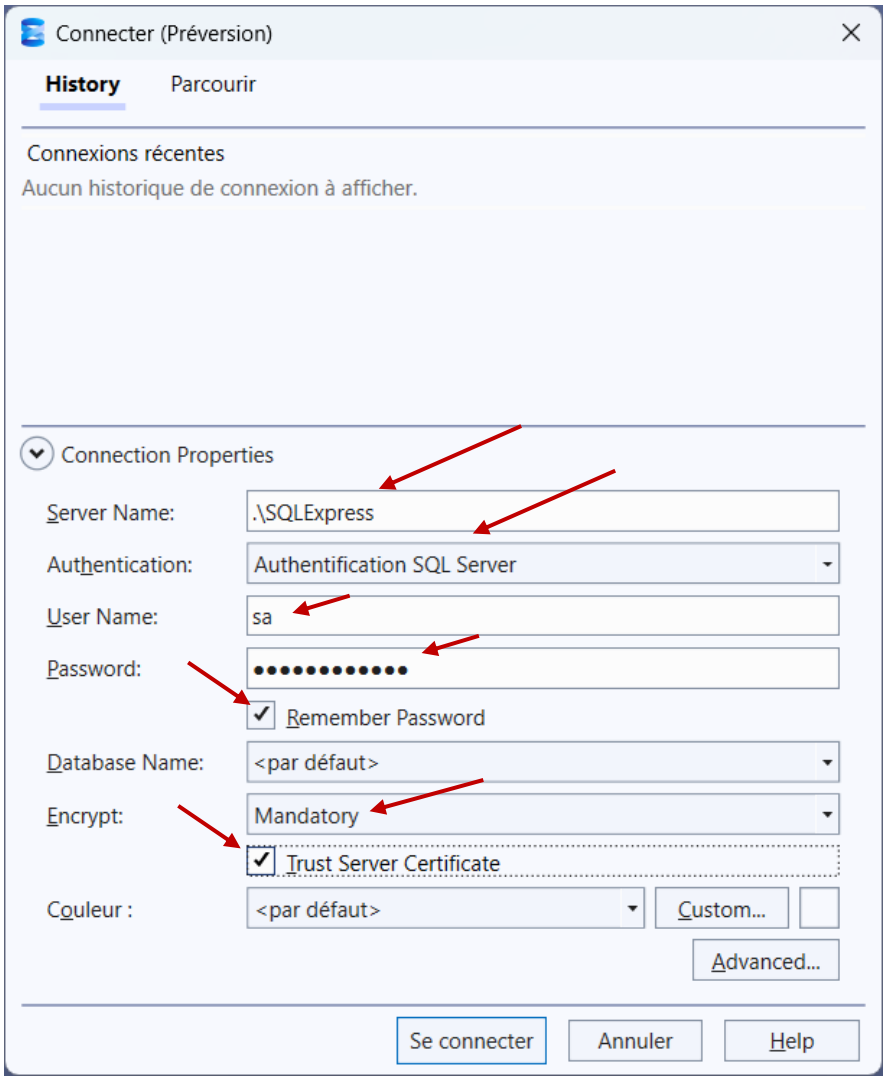
6.2.2 Installation de l'interface de gestion

Pour configurer, gérer et administrer les composants de SQL Server, nous utiliserons l'application Microsoft SQL Server Management Studio. Celle-ci est téléchargeable gratuitement :
https://learn.microsoft.com/fr-fr/ssms/install/install
D'autres alternatives existent, nous pouvons par exemple citer HeidiSQL :
https://www.heidisql.com/download.php

Il n'y a rien de particulier à signaler concernant l'installation de SQL Server Management Studio, il suffit de suivre les étapes proposées.

Lors du lancement de SSMS, au moment de l'écriture de ces notes, le système propose une nouvelle fenêtre de connexion, en voici un aperçu.

Définir les propriétés voulues puis cliquer sur « Se connecter ».



- Dans un cadre d'apprentissage, nous pouvons cocher l'option « Remember Password ».
- **Encrypt : Mandatory**
  La connexion ne se fera **que si elle peut être chiffrée**. Le chiffrement TLS (Transport Layer Security : sécurité de la couche de transport en français) entre la machine cliente (SSMS) et le serveur SQL est activé et obligatoire. Les données échangées sont ainsi protégées contre l'interception par des tiers.
- **En cochant « Trust Server Certificate »** on accepte un certificat non validé officiellement. SSMS fait « confiance » au certificat tel quel, même s'il s'agit d'un certificat auto-signé. Cela ne présente aucun problème dans un cadre pédagogique mais il ne faut pas l'activer dans un environnement de production.

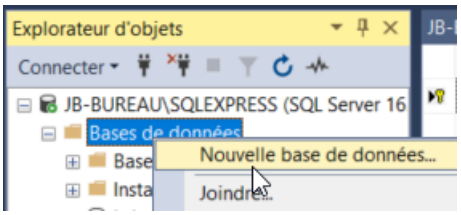
6.3 Créer la base de données et les tables

6.3.1 Créer la base de données et les tables via le concepteur de tables de SQL Server Management Studio (SSMS)

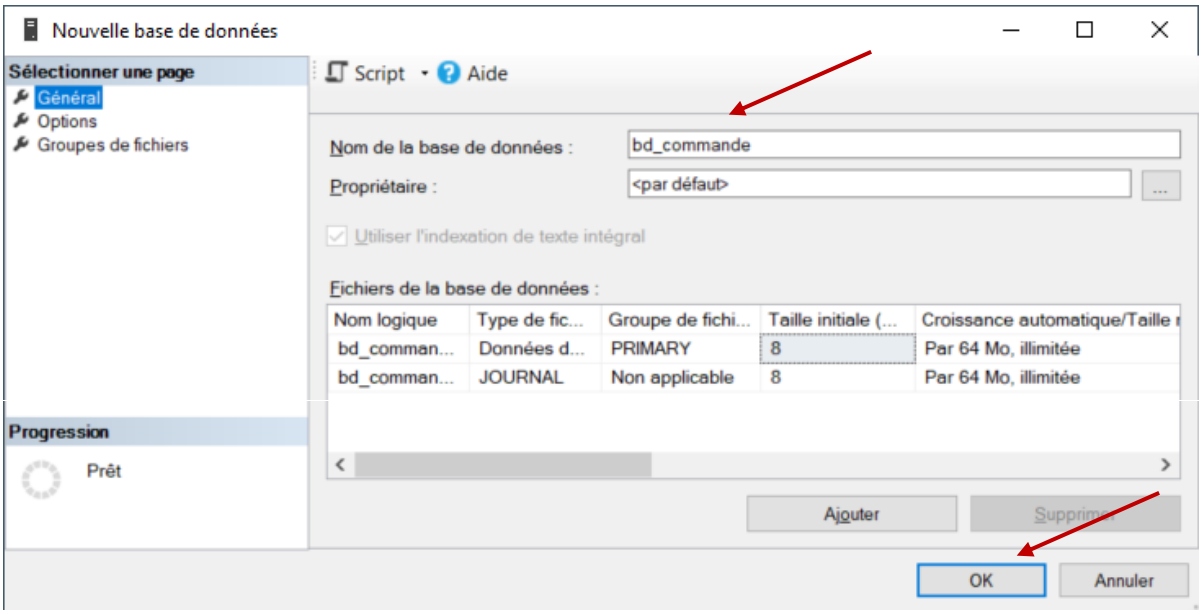
6.3.1.1 Créer le noyau de la base de données

Procédure pour créer une base de données :

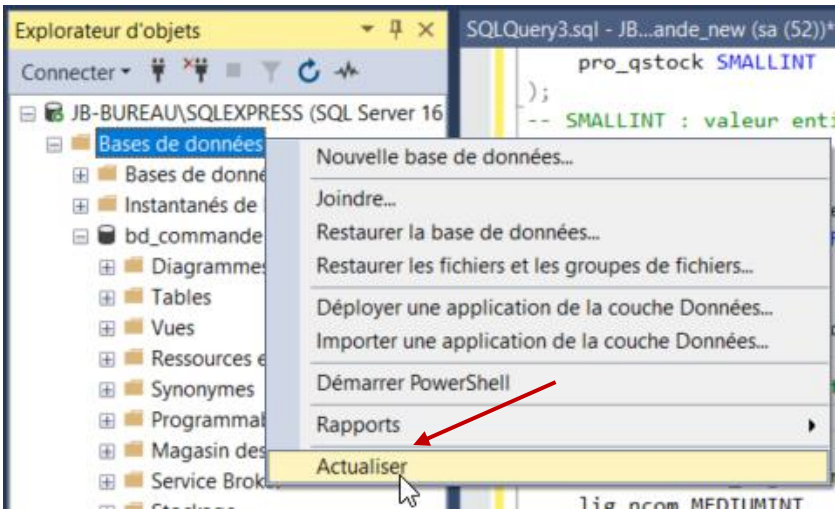
1) Démarrer SSMS (SQL Server Management Studio), cliquer avec le bouton droit de la souris sur « Bases de données » puis choisir « Nouvelle base de données... ».



2) Nommer la base de données. Si nécessaire, définir certaines propriétés notamment présentes dans le volet « Options » puis valider.



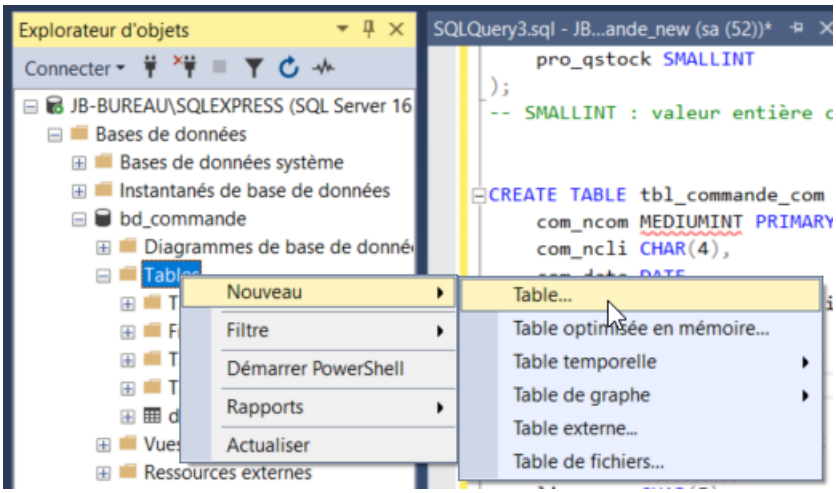
3) La base de données apparaît dans l'Explorateur d'objets. NB : Il faut éventuellement actualiser l'affichage pour que la nouvelle base de données apparaisse.



6.3.1.2 Créer la structure des tables

Procédure pour créer une table :

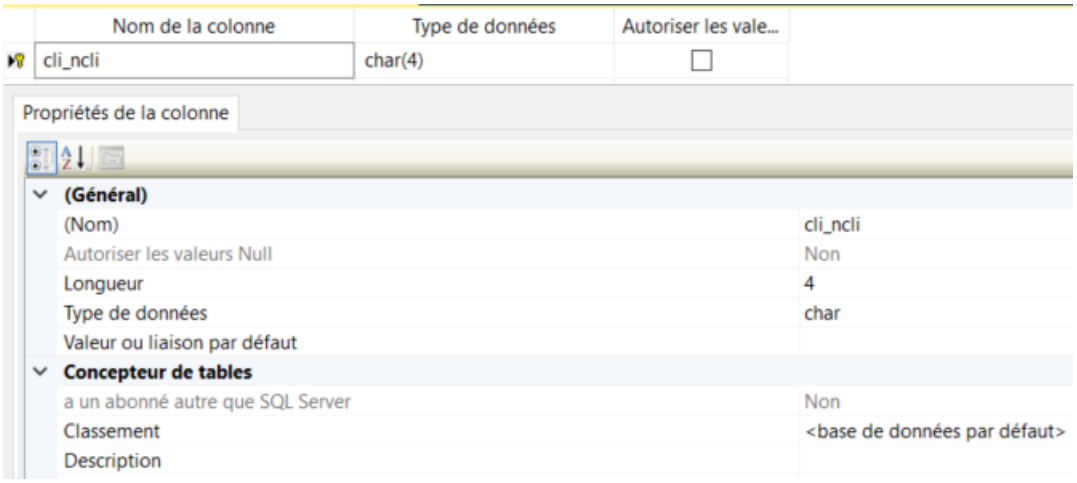
1) Cliquer avec le bouton droit de la souris sur la rubrique « Tables » **sous la base de données voulue** puis choisir « Nouveau / Table... ».



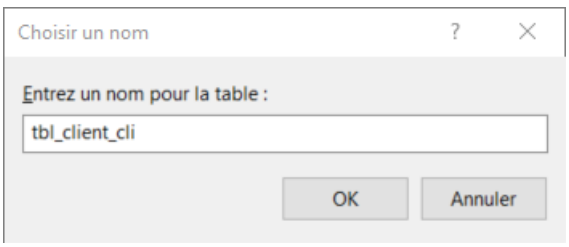
2) Définir les champs : nom, type... Définir la clé primaire.

Nom de la colonne	Type de données	Autoriser les valeurs Null
cli_ncli	char(4)	☐
cli_titre	varchar(10)	☐
cli_nom	varchar(50)	☐
cli_prenom	varchar(50)	☑
cli_rue	varchar(120)	☑
cli_cp	varchar(7)	☑
cli_localite	varchar(50)	☑
cli_cat	char(2)	☑
cli_compte	decimal(8, 2)	☑

Pour chaque champ, des propriétés sont disponibles dans le volet inférieur.



3) Enregistrer la table créée.



En suivant une procédure analogue, créer les tables suivantes :

tbl_produit_pro

Nom de la colonne	Type de données	Autoriser les valeurs Null
pro_npro	char(5)	☐
pro_libelle	varchar(90)	☐
pro_prix	decimal(7, 2)	☑
pro_qstock	smallint	☑

tbl_commande_com

Nom de la colonne	Type de données	Autoriser les valeurs ...
com_ncom	int	☐
com_ncli	char(4)	☑
com_date	date	☑

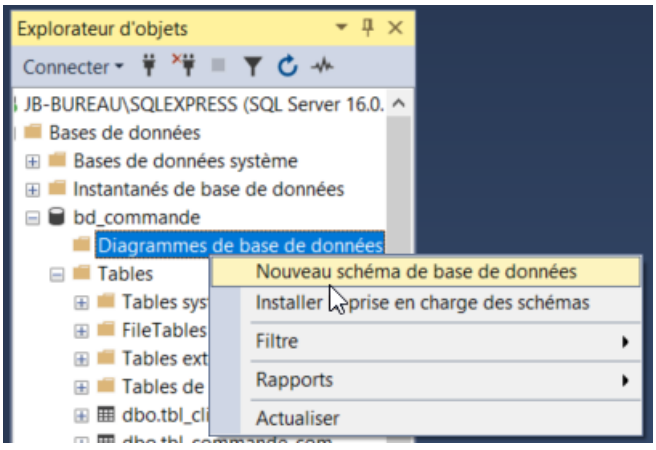
tbl_lignecom_lig

Nom de la colonne	Type de données	Autoriser les valeurs Null
lig_ncom	int	☐
lig_npro	char(5)	☐
lig_prix	decimal(7, 2)	☑
lig_qcom	smallint	☑

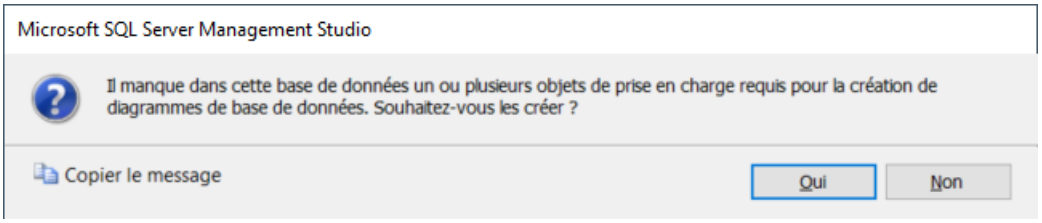
6.3.1.3 Créer des relations entre les tables

Procédure pour créer les relations entre les tables :

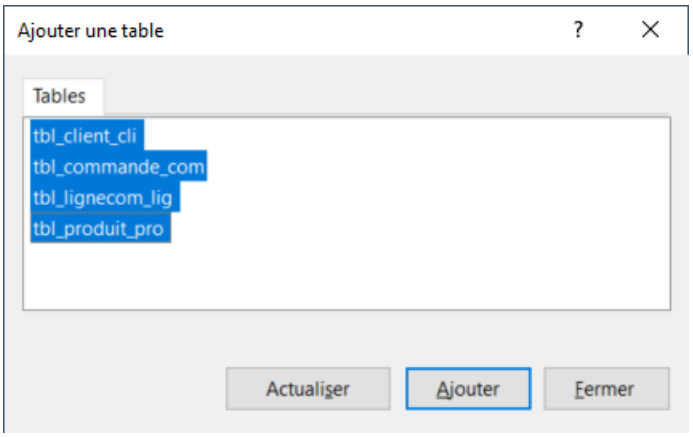
1) Cliquer avec le bouton droit de la souris sur « Diagrammes de base de données » puis sur « Nouveau schéma de base de données ».



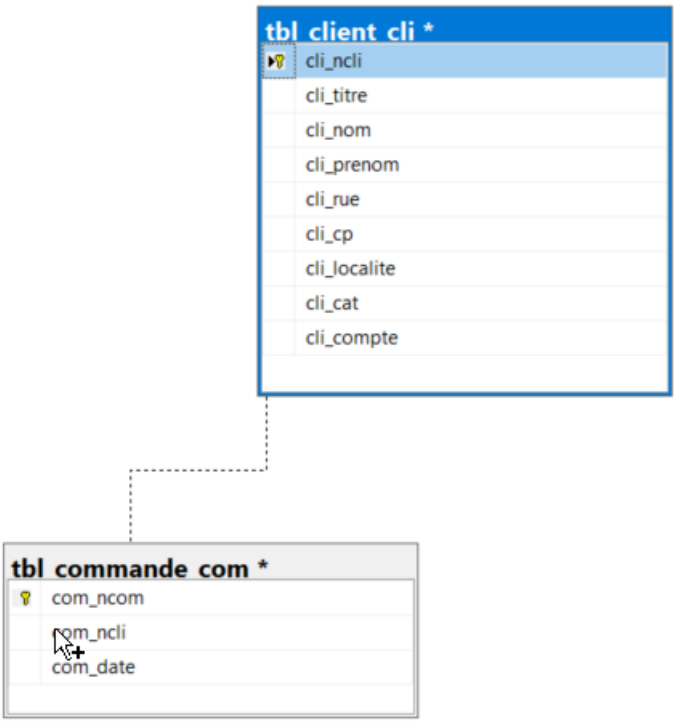
2) Confirmer la création des objets requis pour la prise en charge des diagrammes de base de données.



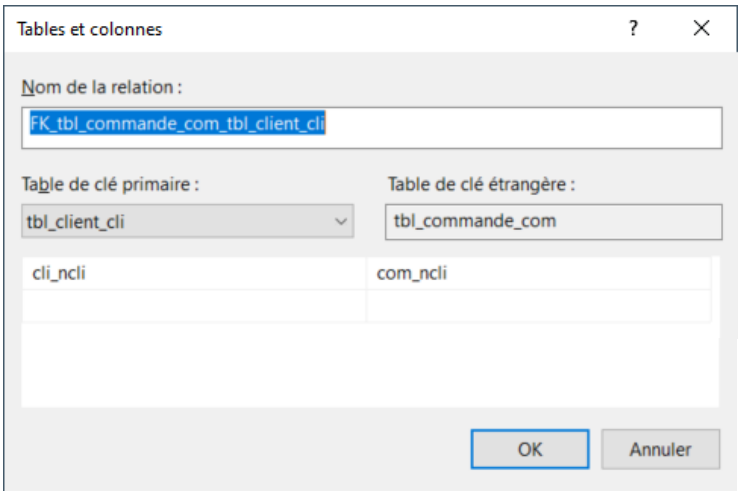
3) Ajouter les tables voulues.



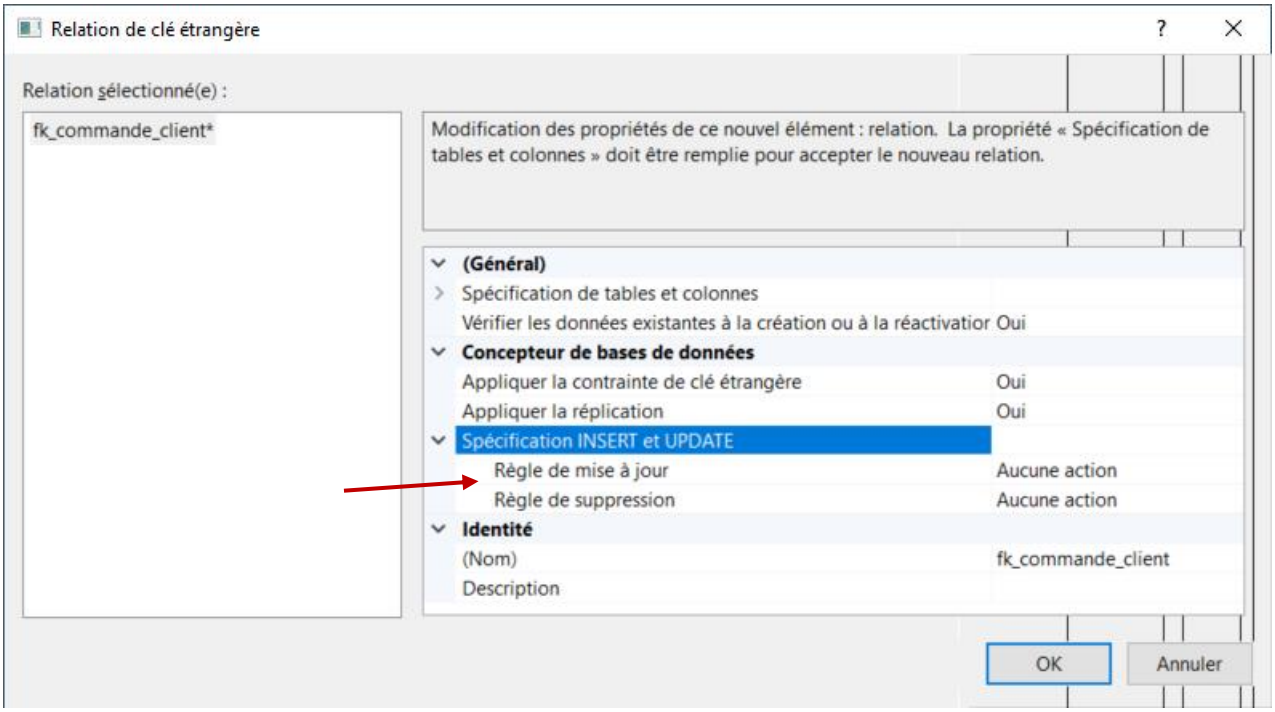
4) Cliquer sur le premier champ concerné par la relation puis tout en maintenant le bouton gauche de la souris enfoncé, glisser vers l'autre champ de la relation.



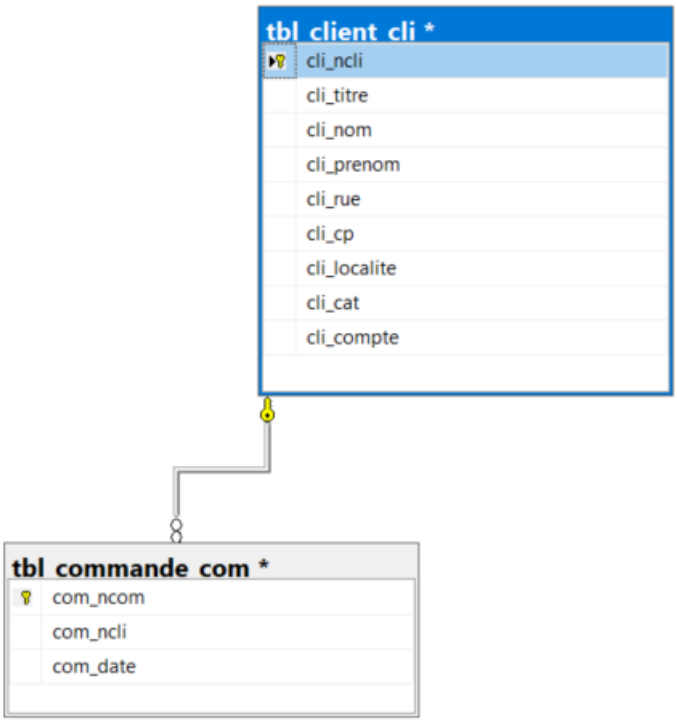
5) Nommer la relation, vérifier les tables et les champs puis valider.



6) Certaines propriétés de la relation peuvent être définies. **Parmi celles-ci, on trouve notamment « Règle de mise à jour » et « Règle de suppression ».**
Valider par OK.



La relation est créée.



7) Créer les autres relations puis enregistrer le diagramme.

6.3.2 Créer la base de données et les tables en utilisant le langage SQL

Dans ce point, le langage SQL est utilisé en tant que langage de définition des données (LDD).

Pour exécuter des requêtes écrites en SQL, il suffit de créer une nouvelle fenêtre en utilisant le bouton « Nouvelle requête ».



NB : Lorsque plusieurs requêtes figurent dans la fenêtre, il suffit de sélectionner celle que l'on souhaite exécuter avant de demander l'exécution sinon le système exécute l'entièreté du code.

6.3.2.1 Créer le noyau de la base de données

```
-- Création du noyau de la base de données
CREATE DATABASE bd_commande;

/* Par défaut, le classement est French_CI_AS
   Convient très bien pour la langue française
   Exemple pour imposer UTF8 :
   CREATE DATABASE bd_commande COLLATE French_100_CI_AS_SC_UTF8;
*/
GO
```

6.3.2.2 Créer la structure des tables et les relations

```
-- Création de la structure des tables
USE bd_commande;

CREATE TABLE tbl_client_cli (
      cli_ncli CHAR(4) PRIMARY KEY,
      cli_titre VARCHAR(10) NOT NULL,
      cli_nom VARCHAR(50) NOT NULL,
      cli_prenom VARCHAR(50),
      cli_rue VARCHAR(120),
      cli_cp VARCHAR(7),
      cli_localite VARCHAR(50),
      cli_cat CHAR(2),
      cli_compte DECIMAL(8,2)
);
/* DECIMAL(8,2) : valeur numérique comportant 8 chiffres significatifs
(précision) dont 2 chiffres après la virgule (échelle) */

CREATE TABLE tbl_produit_pro (
      pro_npro CHAR(5) PRIMARY KEY,
      pro_libelle VARCHAR(90) NOT NULL,
      pro_prix DECIMAL(7,2),
      pro_qstock SMALLINT
);
-- SMALLINT : valeur entière comprise entre -32768 et 32767

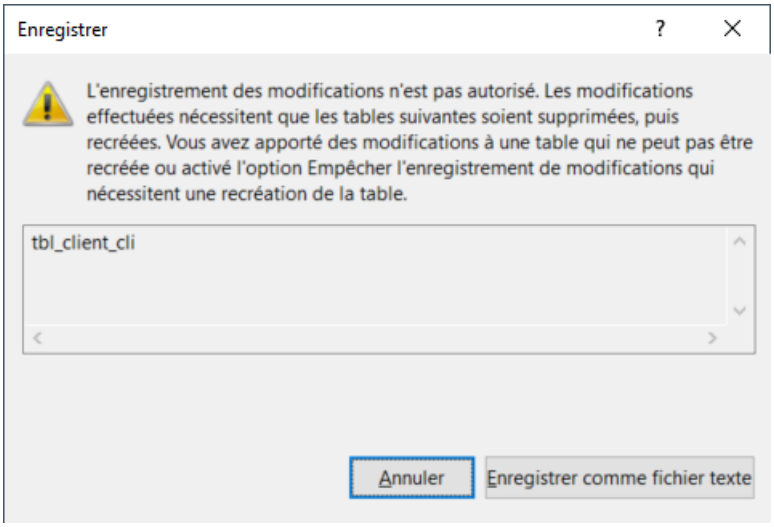
CREATE TABLE tbl_commande_com (
      com_ncom INT PRIMARY KEY,
      com_ncli CHAR(4),
      com_date DATE,
      CONSTRAINT fk_commande_client FOREIGN KEY(com_ncli) REFERENCES
tbl_client_cli(cli_ncli)
);
-- INT : valeur entière comprise entre -2147483648 et 2147483647
```

```
CREATE TABLE tbl_lignecom_lig (
      lig_ncom INT,
      lig_npro CHAR(5),
      lig_prix DECIMAL(7,2),
      lig_qcom SMALLINT,
      PRIMARY KEY (lig_ncom,lig_npro),
      CONSTRAINT fk_lignecom_commande FOREIGN KEY(lig_ncom) REFERENCES
tbl_commande_com(com_ncom),
      CONSTRAINT fk_lignecom_produit FOREIGN KEY(lig_npro) REFERENCES
tbl_produit_pro(pro_npro)
);
```

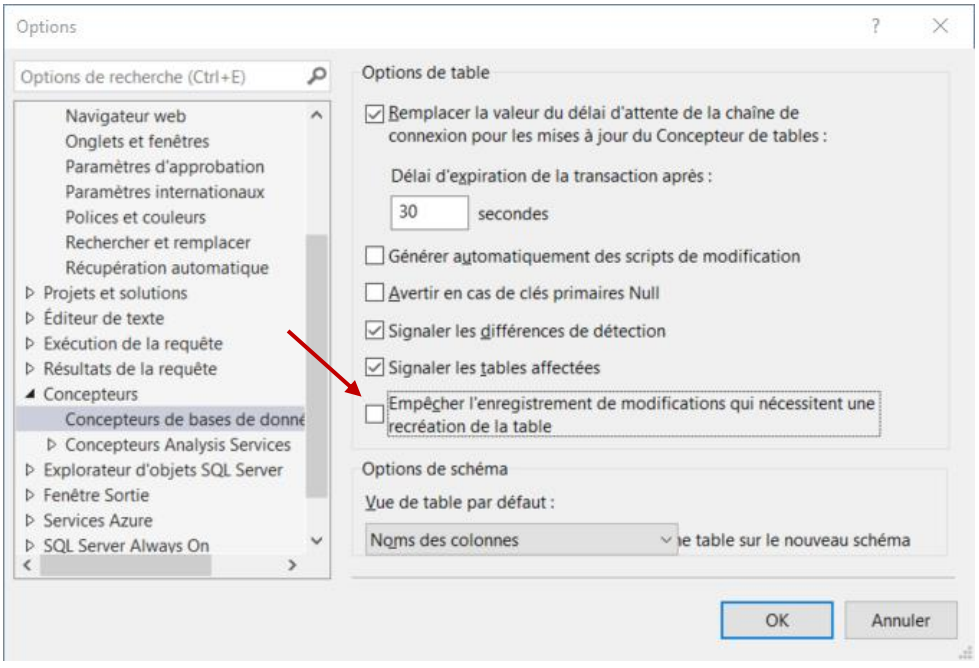
6.3.3 Modifier la définition (structure) d'une table

6.3.3.1 Modifier la structure d'une table via le concepteur de tables de SQL Server Management Studio

Il est important de savoir que par défaut, certaines modifications réalisées via l'interface graphique ne seront pas autorisées. Un message semblable à ce qui suit apparaît alors.



Via « Outils / Options », il est possible de désactiver l'option « Empêcher l'enregistrement de modifications qui nécessitent une recréation de la table ».



Cependant, prudence car :

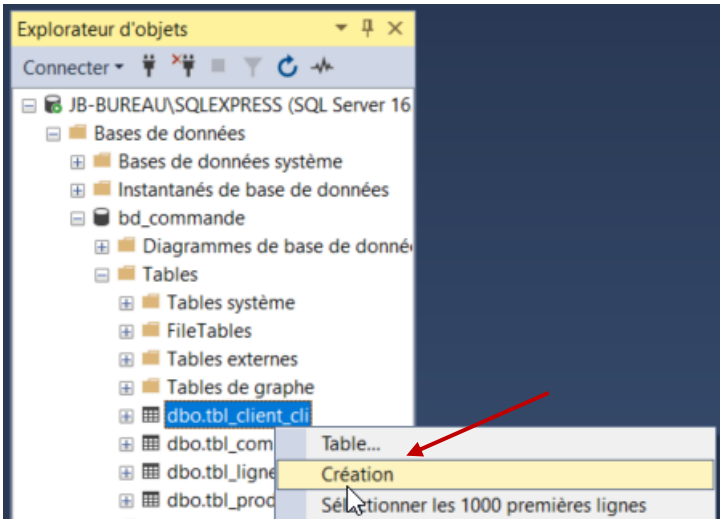
- Perte de données en cas d'échec de l'opération de recréation de la table.
- Problèmes possibles avec les relations, clés étrangères, triggers, etc.
- Perte des autorisations ou contraintes si la recréation est mal gérée.

Dans un environnement de test ou de développement, cette option peut être utilisée avec prudence mais c'est déconseillé dans un environnement de production.

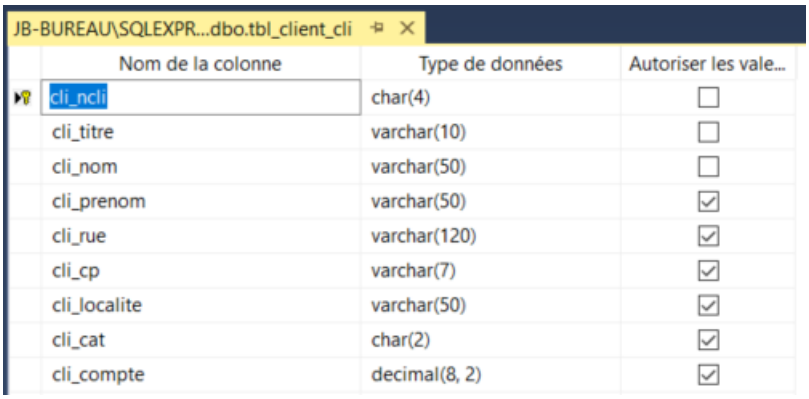
Il est donc parfois (et même souvent) préférable d'utiliser le langage SQL.

Pour modifier la structure d'une table :

1) Effectuer un clic droit sur la table voulue puis sélectionner « Création ».



2) Apporter les modifications voulues puis enregistrer.



6.3.3.2 Modifier la structure d'une table via le langage SQL

Exemples :

```
ALTER TABLE tbl_client_cli
ADD cli_commentaires TEXT;
```

NB : longueur maximale avec le type TEXT : 2^{16} octets

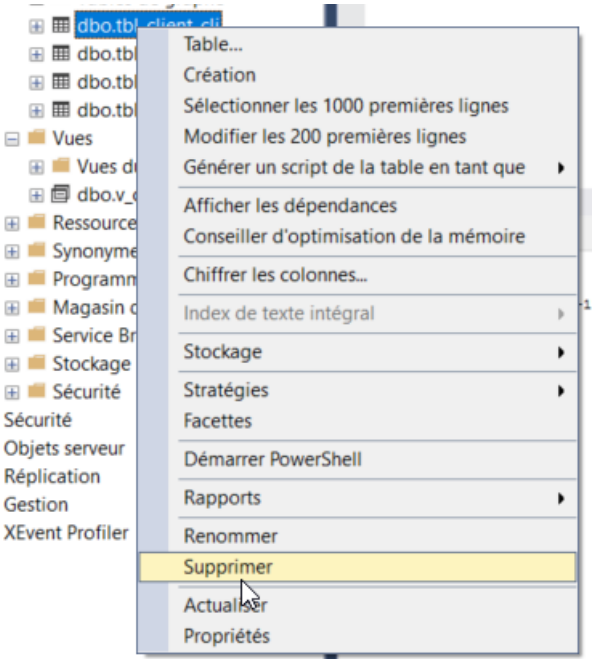
```
ALTER TABLE tbl_client_cli
ALTER COLUMN cli_nom VARCHAR(70);

ALTER TABLE tbl_client_cli
DROP COLUMN cli_commentaires;
```

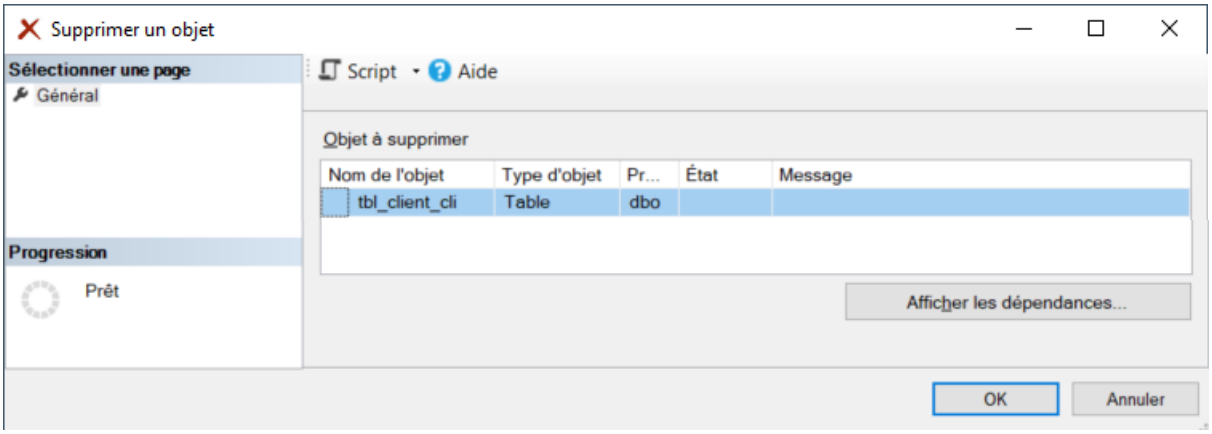
6.3.4 Supprimer une table

6.3.4.1 Supprimer une table via les fonctions graphiques de SQL Server Management Studio

Pour supprimer une table via les fonctions graphiques de SSMS, effectuer un clic droit sur la table voulue puis utiliser la fonction « Supprimer ».



Le système demande de confirmer l'action.



Dans le cadre du contexte de ces notes, le système refusera la suppression car une contrainte d'intégrité référentielle existe entre la table tbl_commande_com et la table tbl_client_cli.



6.3.4.2 Supprimer une table en utilisant le langage SQL

Exemple :

```
DROP TABLE tbl_client_cli;
```

Si on essaie d'exécuter la commande ci-dessus dans le cadre du contexte de ces notes,
le système refuse la suppression car une contrainte d'intégrité référentielle existe entre la table tbl_commande_com et la table tbl_client_cli.

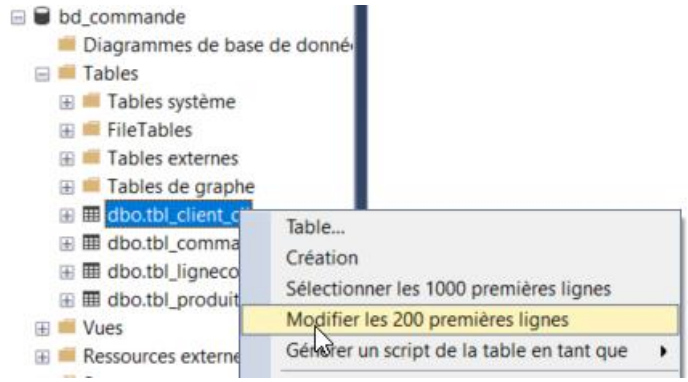
7 Ajouter, mettre à jour, supprimer des enregistrements dans les tables

7.1 Ajouter, mettre à jour, supprimer des enregistrements en utilisant les fonctions graphiques de SQL Server Management Studio

7.1.1 Ajouter des enregistrements dans une table

Pour ajouter des données dans une table via l'interface graphique :

1) Cliquer avec le bouton droit de la souris sur la table voulue puis utiliser « Modifier les 200 premières lignes... ».



2) Encoder les données voulues puis valider.

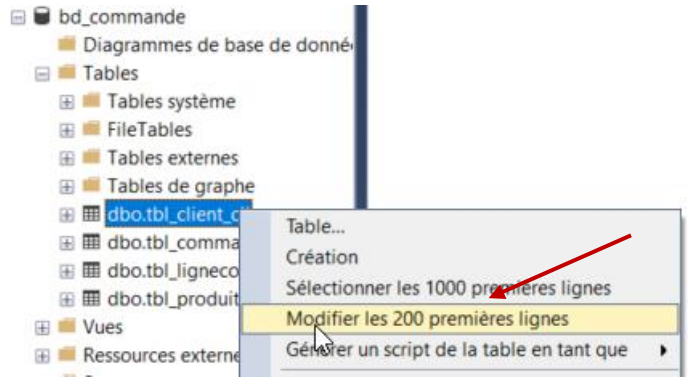
JB-BUREAU\SQLEXPR...dbo.tbl_client_cli

	cli_ncli	cli_titre	cli_nom	cli_prenom	cli_rue	cli_cp	cli_localite	cli_cat	cli_compte
▶	C741	Madame	DUPONT	Julie	NULL	NULL	NULL	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

7.1.2 Modifier des données dans une table

Pour modifier des données dans une table via l'interface graphique :

1) Cliquer avec le bouton droit de la souris sur la table voulue puis utiliser « Modifier les 200 premières lignes... ».



2) Modifier les données voulues puis valider.

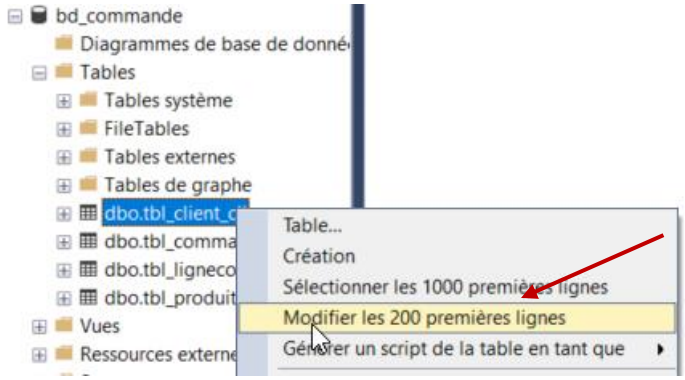
JB-BUREAU\SQLEXPR...dbo.tbl_client_cli

	cli_ncli	cli_titre	cli_nom	cli_prenom	cli_rue	cli_cp	cli_localite	cli_cat	cli_compte
▶	C741	Madame	DUPONT	Juliette	NULL	NULL	NULL	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

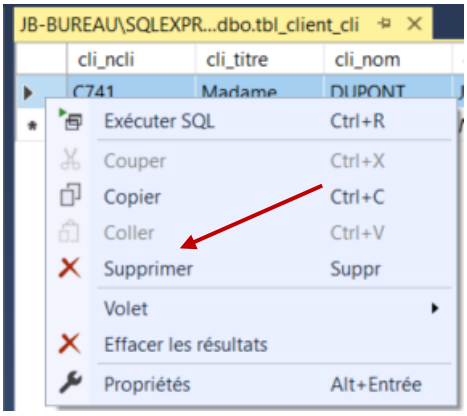
7.1.3 Supprimer des données dans une table

Pour supprimer des données dans une table via l'interface graphique :

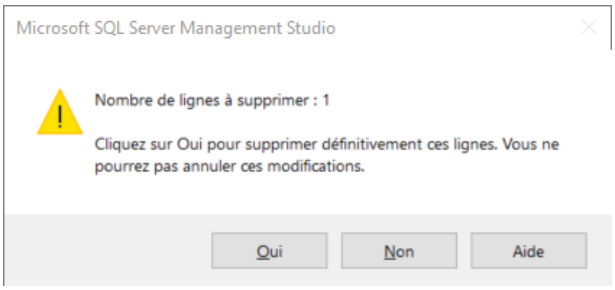
1) Cliquer avec le bouton droit de la souris sur la table voulue puis utiliser « Modifier les 200 premières lignes... ».



2) Sélectionner les lignes voulues puis appuyer sur la touche « Delete » ou utiliser l'option voulue dans le menu contextuel.



3) Le système demande de confirmer la suppression.



7.2 Ajouter, mettre à jour, supprimer des enregistrements en utilisant le langage SQL

Dans ce point, le langage SQL est utilisé en tant que langage de manipulation des données (LMD).

Pour exécuter des requêtes écrites en SQL, il suffit de créer une nouvelle fenêtre en utilisant le bouton « Nouvelle requête ».



NB : Lorsque plusieurs requêtes figurent dans la fenêtre, il suffit de sélectionner celle que l'on souhaite exécuter avant de demander l'exécution sinon le système exécute l'entièreté du code.

7.2.1 La commande INSERT

La commande INSERT permet l'ajout de données dans une table.

```
-- Ajout d'enregistrements dans la table tbl_client_cli

INSERT INTO tbl_client_cli VALUES('B112','Monsieur','Hansenne','Albert','Rue A. Dumont 23','5300','Andenne','C1',1250);
INSERT INTO tbl_client_cli VALUES('C123','Monsieur','Mercier','Jean','Rue Lemaître 25','5000','Namur','C1',-2300);
INSERT INTO tbl_client_cli VALUES('B332','Monsieur','Monti','Paul','Rue Neuve 13','4970','Stavelot','B2',0);
INSERT INTO tbl_client_cli VALUES('F010','Madame','Toussaint','Corine','Rue Godefroid 5','5300','Andenne','C1',0);
INSERT INTO tbl_client_cli VALUES('K111','Madame','Vanbist','Sylvie','Rue Florimont 81','4540','Amay','B1',720);
INSERT INTO tbl_client_cli VALUES('S127','Monsieur','Vanderka','René','Avenue des Roses 3','5000','Namur','C1',-4580);
INSERT INTO tbl_client_cli VALUES('B512','Monsieur','Gillet','Grégoire','Rue de la Gare 14','4000','Liège','B1',-8700);
INSERT INTO tbl_client_cli VALUES('B062','Madame','Goffin','Jacqueline','Rue de la Gare 72','5000','Namur','B2',-3200);
INSERT INTO tbl_client_cli VALUES('C400','Madame','Ferard','Anne','Rue du Tertre 65','5300','Andenne','B2',350);
INSERT INTO tbl_client_cli VALUES('C003','Monsieur','Avron','François','Chemin de la Cure 8','4000','Liège','B1',-1700);
INSERT INTO tbl_client_cli VALUES('K729','Monsieur','Neuman','Herbert','Rue Bransart 40','4000','Liège','C3',0);
INSERT INTO tbl_client_cli VALUES('F011','Monsieur','Poncelet','Fernando','Clos des Erables 17','4000','Liège','B2',0);
INSERT INTO tbl_client_cli VALUES('L422','Madame','Franck','Albertine','Rue de Wépion 60','5000','Namur','C1',0);
INSERT INTO tbl_client_cli VALUES('S712','Monsieur','Guillaume','Albert','Rue du Lac 14a','5100','Wépion','B1',0);
INSERT INTO tbl_client_cli VALUES('D063','Madame','Nadoy','Jacqueline','Boulevard de la Poste 1','4000','Liège','C3',-2250);
INSERT INTO tbl_client_cli VALUES('F400','Monsieur','Jacob','Daniel','Chemin du Moulin 78','3000','Louvain','C2',0);


-- Ajout d'enregistrements dans la table tbl_produit_pro

INSERT INTO tbl_produit_pro VALUES ('CS262','Chevron en sapin 300 x 4 x 6 cm',5.50,45);
INSERT INTO tbl_produit_pro VALUES ('CS264','Chevron en sapin 300 x 6 x 6 cm',6.80,2690);
INSERT INTO tbl_produit_pro VALUES ('CS464','Chevron en sapin 400 x 4 x 6 cm',6.00,450);
INSERT INTO tbl_produit_pro VALUES ('PA45','Pointes en acier 45 (1 kg)',4.00,580);
INSERT INTO tbl_produit_pro VALUES ('PA60','Pointes en acier 60 (1 kg)',5.00,134);
INSERT INTO tbl_produit_pro VALUES ('PH222','Planche en hêtre 200 x 2 x 6 cm',12.00,782);
INSERT INTO tbl_produit_pro VALUES ('PS222','Planche en hêtre 200 x 3 x 6 cm',15.00,1220);


-- Ajout d'enregistrements dans la table tbl_commande_com

INSERT INTO tbl_commande_com VALUES (178,'K111','2025-06-12');
INSERT INTO tbl_commande_com VALUES (179,'C400','2025-06-12');
INSERT INTO tbl_commande_com VALUES (182,'S127','2025-06-14');
INSERT INTO tbl_commande_com VALUES (184,'C400','2025-06-27');
INSERT INTO tbl_commande_com VALUES (185,'F011','2025-07-02');
```

```
INSERT INTO tbl_commande_com VALUES (186,'C400','2025-07-02');
INSERT INTO tbl_commande_com VALUES (188,'B512','2025-07-03');


-- Ajout d'enregistrements dans la table tbl_lignecom_lig

INSERT INTO tbl_lignecom_lig VALUES (178,'CS464',6.00,25);
INSERT INTO tbl_lignecom_lig VALUES (179,'PA60',5.00,20);
INSERT INTO tbl_lignecom_lig VALUES (179,'CS262',5.50,60);
INSERT INTO tbl_lignecom_lig VALUES (182,'PA60',5.00,30);
INSERT INTO tbl_lignecom_lig VALUES (184,'CS464',6.00,120);
INSERT INTO tbl_lignecom_lig VALUES (184,'PA45',4.00,20);
INSERT INTO tbl_lignecom_lig VALUES (185,'PA60',5.00,15);
INSERT INTO tbl_lignecom_lig VALUES (185,'PS222',15.00,600);
INSERT INTO tbl_lignecom_lig VALUES (185,'CS464',6.00,260);
INSERT INTO tbl_lignecom_lig VALUES (186,'PA45',4.00,3);
INSERT INTO tbl_lignecom_lig VALUES (188,'PA60',5.00,70);
INSERT INTO tbl_lignecom_lig VALUES (188,'PH222',12.00,92);
INSERT INTO tbl_lignecom_lig VALUES (188,'CS464',6.00,180);
INSERT INTO tbl_lignecom_lig VALUES (188,'PA45',4.00,22);
```

7.2.2 La commande UPDATE

La commande UPDATE permet de modifier le contenu d'enregistrements existants.

Exemples :

```
UPDATE tbl_client_cli
SET cli_cat='C1'
WHERE cli_ncli='F011';

UPDATE tbl_client_cli
SET cli_cat='B2'
WHERE cli_ncli='F011';
```

7.2.3 La commande DELETE

La commande DELETE permet de supprimer certains enregistrements.

```
DELETE
FROM tbl_client_cli
WHERE cli_ncli='C400';
```

NB : Les contraintes d'intégrité référentielles vont jouer leur rôle.

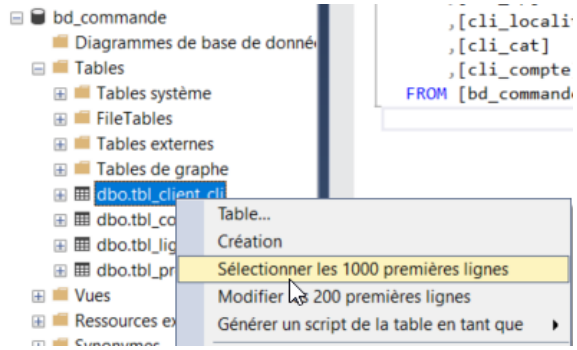
8 Consulter les données

8.1 Consulter les données en utilisant l'interface graphique de SSMS

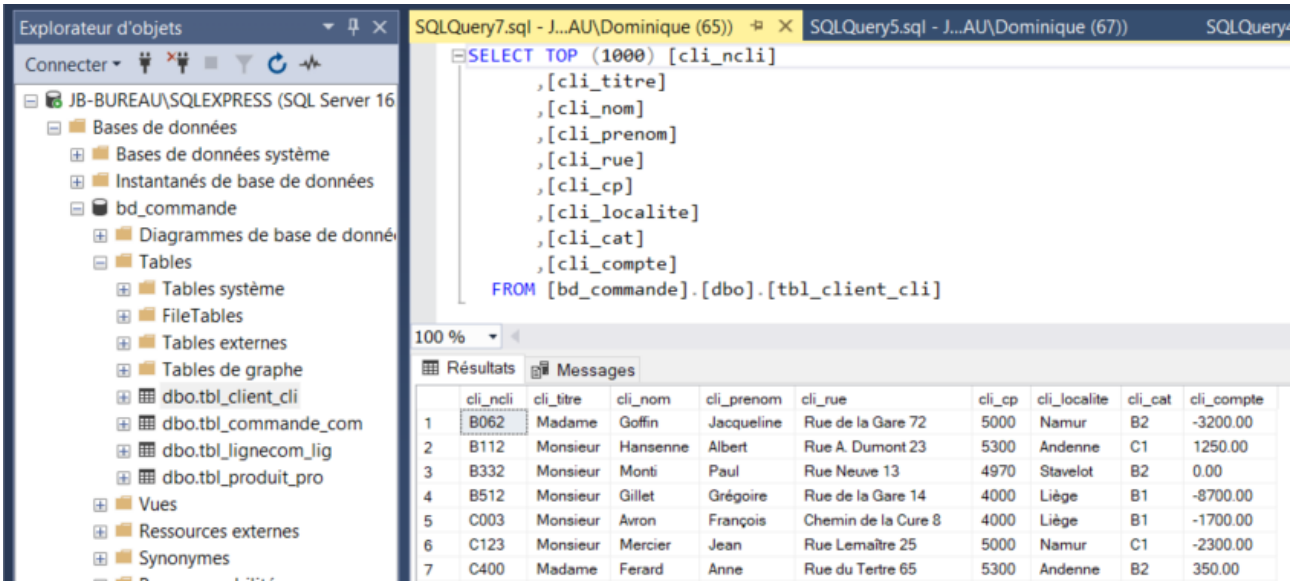
8.1.1 Afficher le contenu d'une table

Procédure pour afficher le contenu d'une table :

1) Effectuer un clic droit sur la table voulue puis cliquer sur « Sélectionner les 1000 premières lignes ».



2) Le système affiche le résultat voulu.



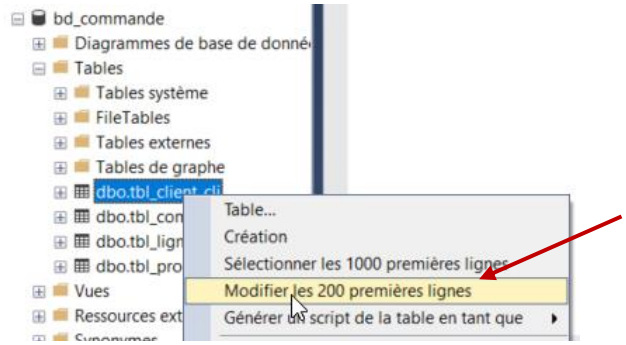
...

8.1.2 Extraire des données d'une table avec définition de critères

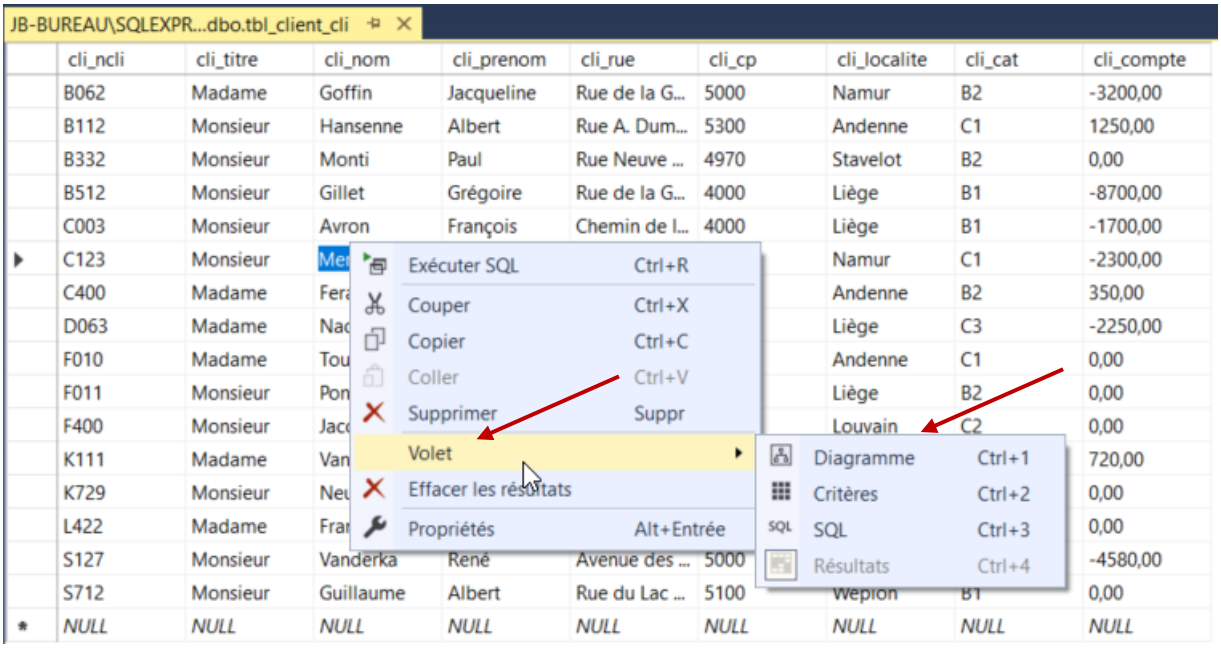
Objectif : Afficher tous les clients habitant Andenne. Le résultat sera trié en fonction du nom de famille.

Procédure :

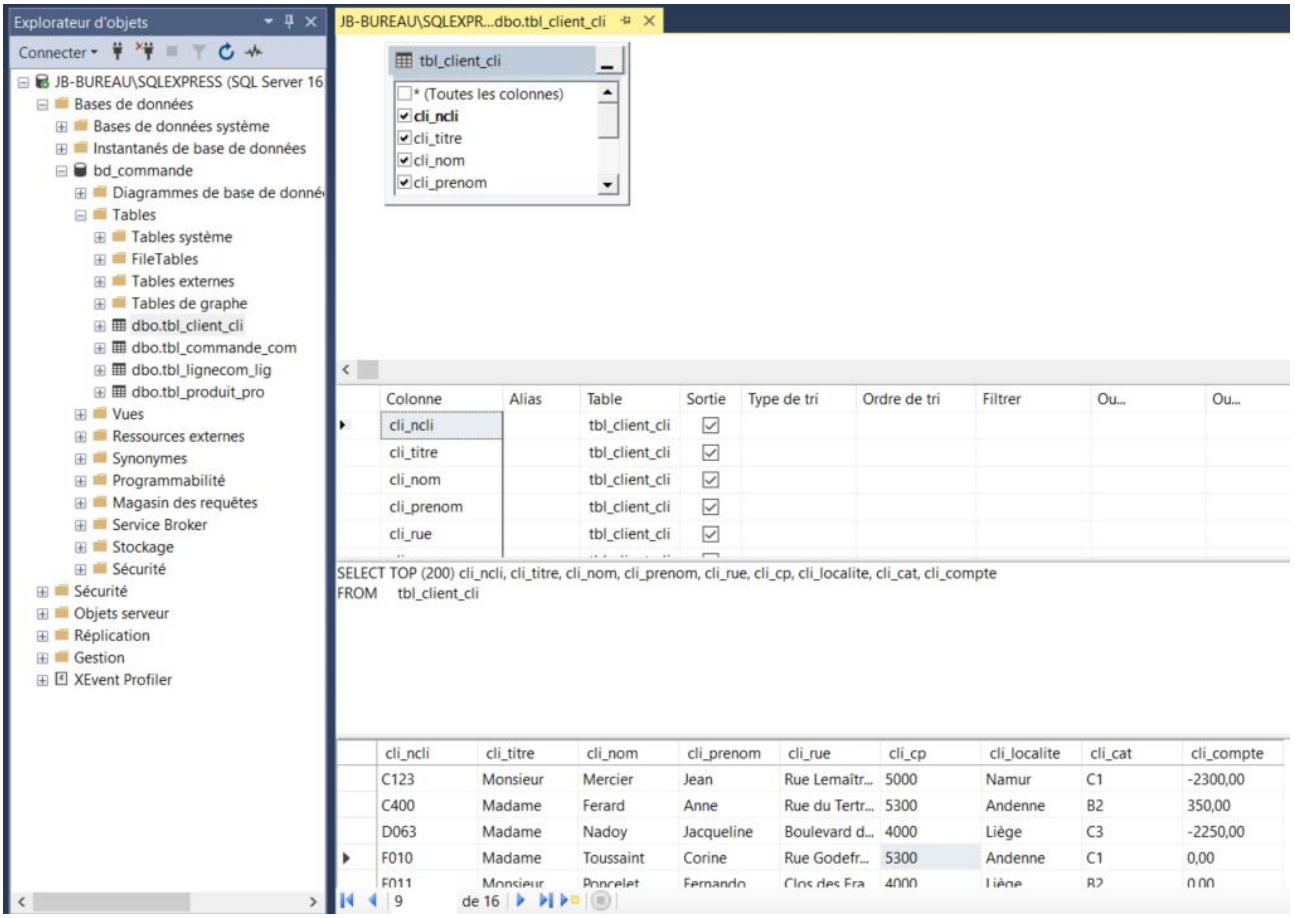
1) Effectuer un clic droit sur la table concernée puis cliquer sur « Modifier les 200 premières lignes ».
NB : C'est contre-intuitif car l'objectif n'est pas de modifier des données mais c'est actuellement le chemin le plus intéressant pour obtenir les 4 volets souhaités.



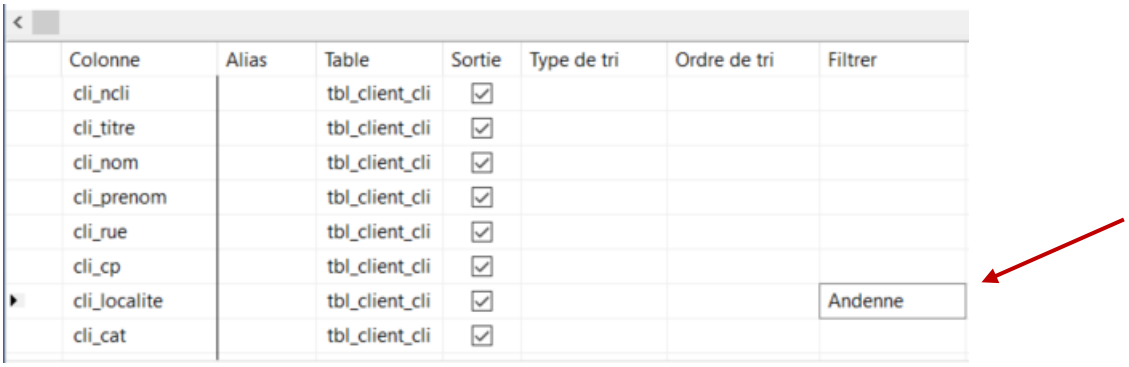
2) Effectuer un clic droit dans la feuille de données, survoler « Volet » puis activer « Diagramme », « Critères » et « SQL » (le volet « Résultats » est coché par défaut). L'action devra se faire en plusieurs fois.



3) Une fenêtre semblable à ce qui suit est obtenue. NB : Dans la requête, la clause TOP (200) peut être supprimée si nécessaire.



4) Définir le critère de sélection voulu.

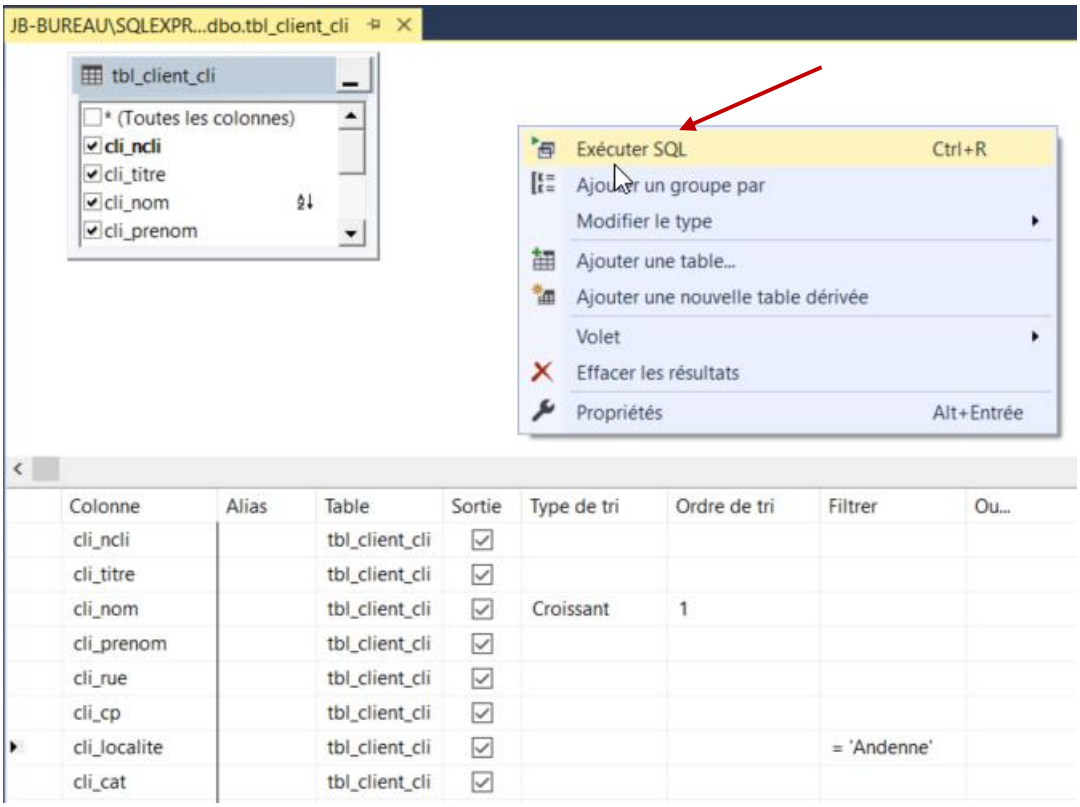


Colonne	Alias	Table	Sortie	Type de tri	Ordre de tri	Filtrer
cli_ncli		tbl_client_cli	☑			
cli_titre		tbl_client_cli	☑			
cli_nom		tbl_client_cli	☑			
cli_prenom		tbl_client_cli	☑			
cli_rue		tbl_client_cli	☑			
cli_cp		tbl_client_cli	☑			
cli_localite		tbl_client_cli	☑			Andenne
cli_cat		tbl_client_cli	☑			

5) Définir le critère de tri.

Colonne	Alias	Table	Sortie	Type de tri	Ordre de tri	Filtrer
cli_ncli		tbl_client_cli	☑			
cli_titre		tbl_client_cli	☑			
cli_nom		tbl_client_cli	☑	Croissant	1	
cli_prenom		tbl_client_cli	☑			
cli_rue		tbl_client_cli	☑			
cli_cp		tbl_client_cli	☑			
cli_localite		tbl_client_cli	☑			= 'Andenne'
cli_cat		tbl_client_cli	☑			

6) Exécuter la requête (via le menu contextuel par exemple).



Colonne	Alias	Table	Sortie	Type de tri	Ordre de tri	Filtrer	Ou...
cli_ncli		tbl_client_cli	☑				
cli_titre		tbl_client_cli	☑				
cli_nom		tbl_client_cli	☑	Croissant	1		
cli_prenom		tbl_client_cli	☑				
cli_rue		tbl_client_cli	☑				
cli_cp		tbl_client_cli	☑				
cli_localite		tbl_client_cli	☑			= 'Andenne'	
cli_cat		tbl_client_cli	☑				

Résultat obtenu :

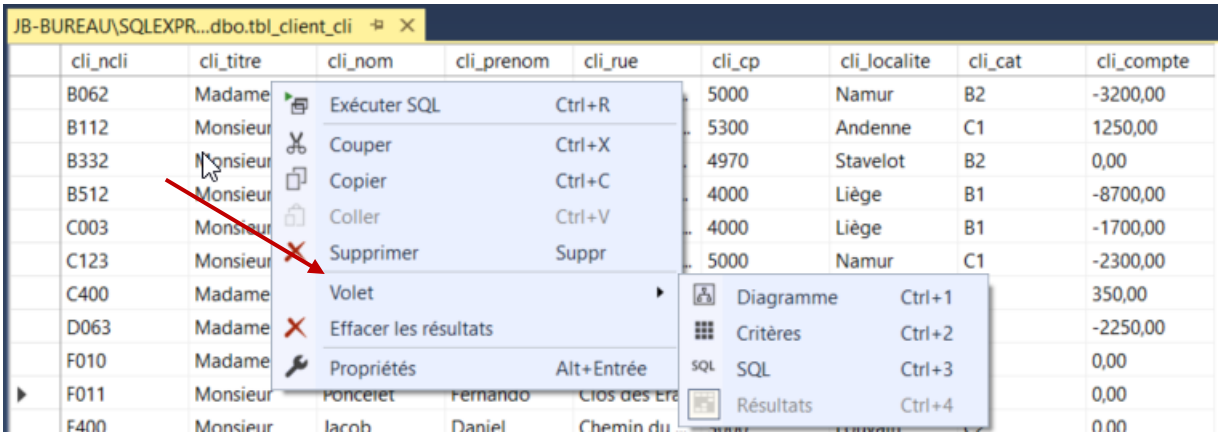
cli_ncli	cli_titre	cli_nom	cli_prenom	cli_rue	cli_cp	cli_localite	cli_cat	cli_compte
C400	Madame	Ferard	Anne	Rue du Tertr...	5300	Andenne	B2	350,00
B112	Monsieur	Hansenne	Albert	Rue A. Dum...	5300	Andenne	C1	1250,00
F010	Madame	Toussaint	Corine	Rue Godefr...	5300	Andenne	C1	0,00

8.1.3 Extraire des données provenant de différentes tables avec définition de critères

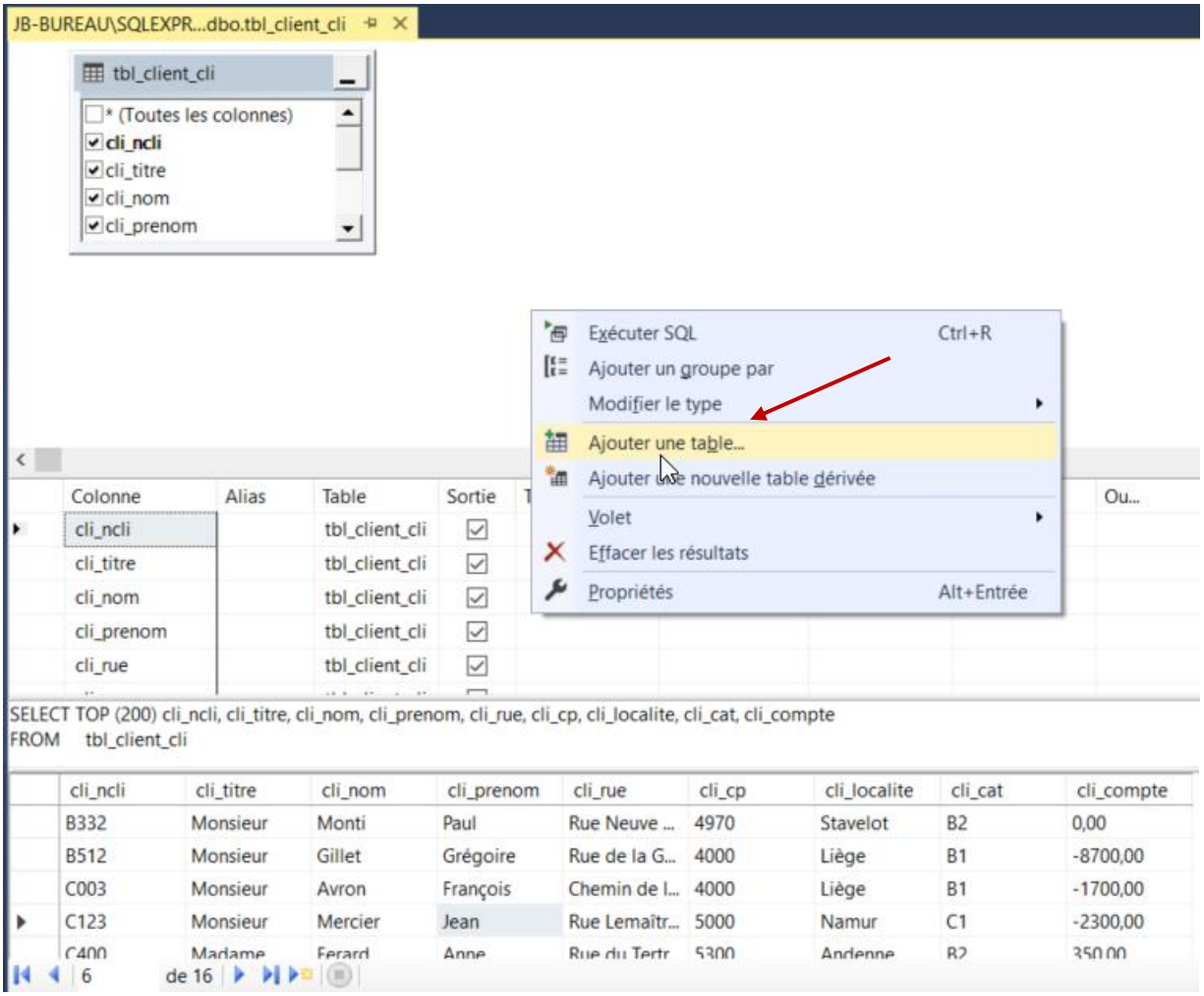
Objectif : Afficher les clients avec en vis-à-vis les commandes passées. Les données seront triées en fonction du champ NCLI. Il s'agit ici d'une requête de type jointure.

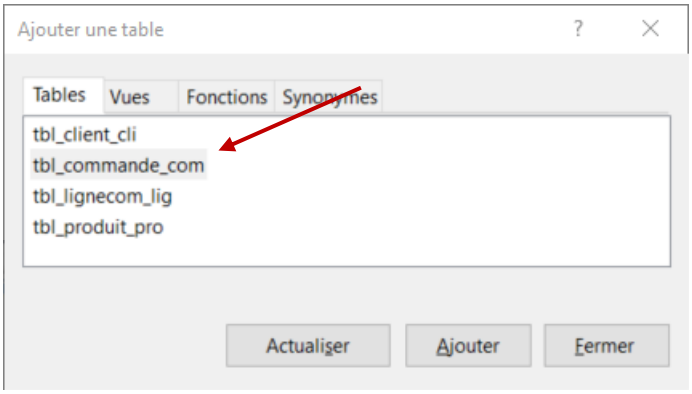
1) Effectuer un clic droit sur une des tables concernées (tbl_client_cli par exemple) puis cliquer « Modifier les 200 premières lignes ». NB : C'est contre-intuitif car l'objectif n'est pas de modifier des données mais c'est actuellement le chemin le plus intéressant pour obtenir les 4 volets souhaités.

2) Effectuer un clic droit dans la feuille de données, survoler « Volet » puis activer « Diagramme », « Critères » et « SQL » (le volet « Résultats » est coché par défaut). L'action devra se faire en plusieurs fois.

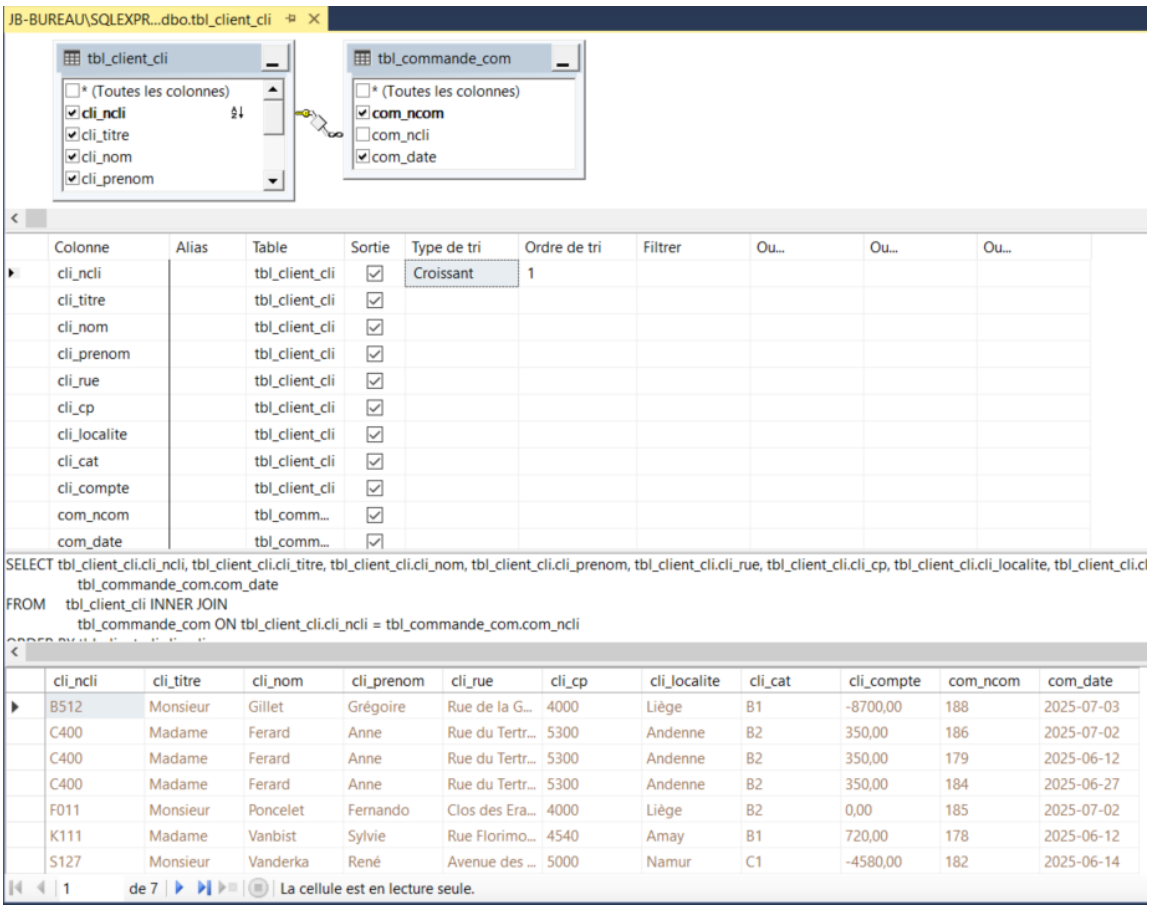


3) Une fenêtre semblable à ce qui suit est obtenue. NB : Dans la requête, la clause TOP (200) peut être supprimée si nécessaire. Ajouter la table tbl_commande_com.





4) On peut constater que la relation définie lors de la création des tables est bien présente. Elle permet de définir la condition de jointure. Cocher les champs voulus puis exécuter.

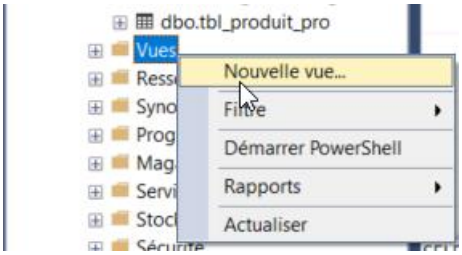


8.1.4 Enregistrer une requête en tant que vue

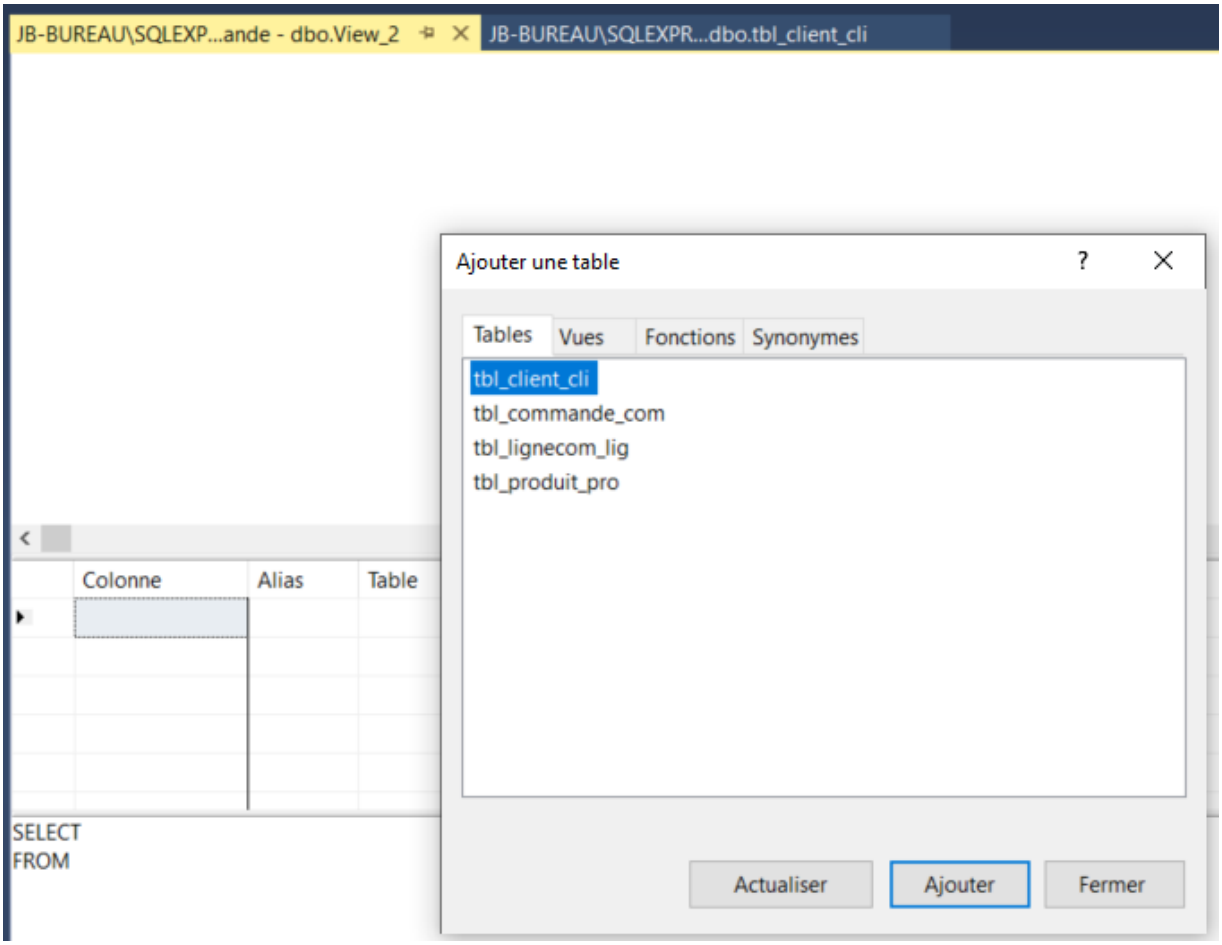
Une vue est une requête enregistrée dans la base de données sous la forme d'un objet.
Elle permet de présenter les données issues d'une ou plusieurs tables comme si elles formaient une table unique, sans dupliquer les données.

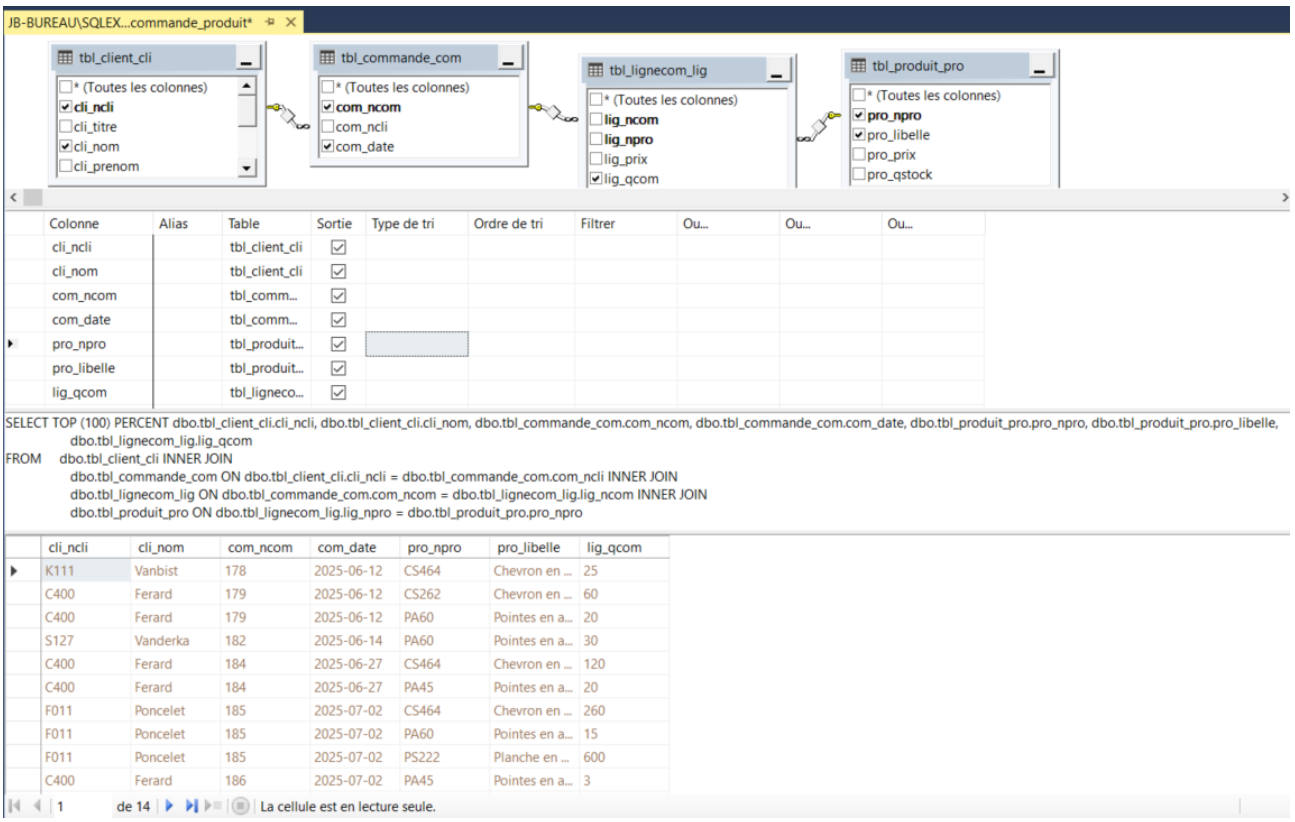
Procédure pour créer une vue :

1) Effectuer un clic droit sur « Vues » puis utiliser « Nouvelle vue ».

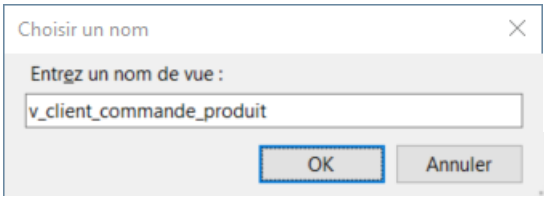


2) Le système propose alors d'ajouter les tables voulues...
Il suffit de suivre une procédure analogue à ce qui a été vu précédemment.

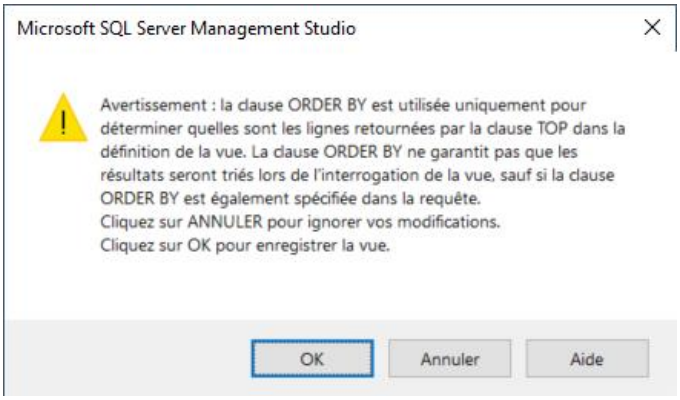




Une fois la vue créée et testée, elle pourra être enregistrée.



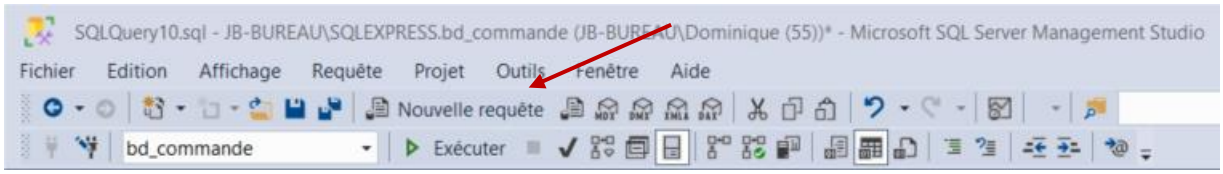
Note :
Il est conseillé de ne pas définir de critère de tri dans la vue sauf si cela peut aider lors du test de celle-ci. Comme nous pouvons le voir ci-après, cette clause ORDER BY dans la vue ne garantit pas que le résultat sera trié lors de l'appel de cette vue. Il faut définir le tri dans la requête qui appelle la vue.



NB : Il est généralement conseillé de créer la vue via du code.

8.2 Consulter les données en utilisant le langage SQL

Pour exécuter des requêtes écrites en SQL, il suffit de créer une nouvelle fenêtre en utilisant le bouton « Nouvelle requête ».



NB : Lorsque plusieurs requêtes figurent dans la fenêtre, il suffit de sélectionner celle que l'on souhaite exécuter avant de demander l'exécution sinon le système exécute l'entièreté du code.

8.2.1 La commande SELECT

La commande SELECT permet de consulter le contenu d'une base de données, d'extraire des informations. Voici quelques exemples de requêtes.

8.2.1.1 Créer des requêtes portant sur une seule table

1) Afficher les caractéristiques de tous les produits.

```
SELECT *
FROM tbl_produit_pro;
```

2) Afficher les localités des clients (sans doubles).

```
SELECT DISTINCT cli_localite
FROM tbl_client_cli
ORDER BY cli_localite;
```

3) Afficher le numéro, le nom et le prénom de tous les clients (le nom et le prénom doivent être concaténés).

```
SELECT cli_ncli, CONCAT(cli_nom,' ',cli_prenom) AS cli_nom_prenom
FROM tbl_client_cli;
```

4) Afficher le numéro, le nom et la localité des clients qui habitent à Liège ou à Namur.

```
SELECT cli_ncli, cli_nom, cli_localite
FROM tbl_client_cli
WHERE cli_localite IN ('Liège','Namur');
```

5) Afficher le numéro des clients dont le compte est compris entre -5000 et 0.

```
SELECT cli_ncli, cli_compte
FROM tbl_client_cli
WHERE cli_compte BETWEEN -5000 and 0;
```

6) Afficher les caractéristiques des produits en « sapin ».

```
SELECT *
FROM tbl_produit_pro
WHERE pro_libelle LIKE '%sapin%';

-- OU  (important si le SGBD effectue une différenciation sur la casse)

SELECT *
FROM tbl_produit_pro
WHERE LOWER(pro_libelle) LIKE '%sapin%';
```

7) Afficher le total, le minimum, la moyenne et le maximum des comptes des clients.

```
SELECT SUM(cli_compte) as somme_compte , MIN(cli_compte) as min_compte,
ROUND(AVG(cli_compte),2) AS moyenne_compte,MAX(cli_compte) as max_compte
FROM tbl_client_cli;
```

8) Afficher le numéro et le nom des clients qui n'habitent ni à Namur, ni à Wépion ni à Amay.

```
SELECT cli_ncli, cli_nom, cli_localite
FROM tbl_client_cli
WHERE cli_localite NOT IN ('Namur','Wépion','Amay');
```

9) Afficher les clients qui appartiennent à la catégorie C1 et/ou qui habitent à Namur.

```
SELECT *
FROM tbl_client_cli
WHERE cli_cat='C1' OR cli_localite='Namur';
```

10) Parmi les clients de catégorie C1, afficher ceux qui habitent à Namur.

```
SELECT *
FROM tbl_client_cli
WHERE cli_cat='C1' AND cli_localite='Namur';
```

11) Afficher les clients qui répondent à au moins une des conditions suivantes :
- être de catégorie B1 ou C1
- habiter à Liège ou à Namur.

```
SELECT *
FROM tbl_client_cli
WHERE cli_cat IN ('B1','C1') OR cli_localite IN ('Liège','Namur');
```

8.2.1.2 Créer des requêtes avec opération de regroupement

Base de données concernée : bd_commande

1) Compter le nombre de clients par catégorie.

```
SELECT cli_cat AS 'Catégorie', COUNT(*) AS 'Nombre de clients'
FROM tbl_client_cli
GROUP BY cli_cat;
```

2) Compter le nombre de clients par localité.

```
SELECT cli_localite AS 'Localité', COUNT(*) AS 'Nombre de clients'
FROM tbl_client_cli
GROUP BY cli_localite;
```

3) Afficher les localités qui comptent plus de 3 clients.

```
SELECT cli_localite AS 'Localité', COUNT(*) AS 'Nombre de clients'
FROM tbl_client_cli
GROUP BY cli_localite
HAVING COUNT(*)>3;
```

4) Afficher le nombre de clients.

```
SELECT COUNT(*) AS 'Nombre de clients'
FROM tbl_client_cli;
```

8.2.1.3 Créer des requêtes avec sous-requêtes

Base de données concernée : bd_commande

NB : Certaines requêtes figurant ci-après peuvent également être formulées sous forme de jointures.

1) Afficher les clients (NCLI, NOM, PRENOM, LOCALITE) qui habitent la même localité que le(la) client(e) C400.

```
SELECT cli_ncli, cli_nom, cli_prenom, cli_localite
FROM tbl_client_cli
WHERE cli_localite =  (SELECT cli_localite
                      FROM tbl_client_cli
                      WHERE cli_ncli='C400');
```

2) Afficher les produits dont le prix est supérieur à la moyenne du prix de tous les produits.

```
SELECT pro_npro, pro_libelle, pro_prix
FROM tbl_produit_pro
WHERE pro_prix >  (SELECT AVG(pro_prix)
                  FROM tbl_produit_pro);
```

3) Elaborer une requête permettant de sélectionner tous les clients (NCLI, NOM) n'ayant passé aucune commande en 2022.

```
SELECT cli_ncli, cli_nom
FROM tbl_client_cli
WHERE cli_ncli NOT IN  (SELECT com_ncli
                       FROM tbl_commande_com
                       WHERE YEAR(com_date)=2025);
```

4) Afficher le numéro des clients qui ont passé des commandes qui spécifient le produit CS464.

```
SELECT com_ncli
FROM tbl_commande_com
WHERE com_ncom IN  (SELECT lig_ncom
                   FROM tbl_lignecom_lig
                   WHERE lig_npro='CS464');
```

5) Afficher le numéro, le nom et le prénom des clients qui ont commandé le produit PA45.

```
SELECT cli_ncli, cli_nom, cli_prenom
FROM tbl_client_cli
WHERE cli_ncli IN  (SELECT com_ncli
                   FROM tbl_commande_com
                   WHERE com_ncom IN  (SELECT lig_ncom
                                       FROM tbl_lignecom_lig
                                       WHERE lig_npro='PA45'));
```

6) Afficher les numéros des commandes qui ont été passées par un client habitant Amay ou Andenne.

```
SELECT com_ncom
FROM tbl_commande_com
WHERE com_ncli IN  (SELECT cli_ncli
                   FROM tbl_client_cli
                   WHERE cli_localite IN ('Amay','Andenne'));
```

7) Afficher les numéros de tous les bons de commande sauf ceux qui ont été passés par des clients d'Andenne.

```
SELECT com_ncom
FROM tbl_commande_com
WHERE com_ncli NOT IN  (SELECT cli_ncli
                        FROM tbl_client_cli
                        WHERE cli_localite='Andenne');
```

8) Afficher l'identifiant des client(s) ayant commandé au moins deux fois le produit PA45

```
SELECT com_ncli
FROM tbl_commande_com
WHERE com_ncom IN  (SELECT lig_ncom
                    FROM tbl_lignecom_lig
                    WHERE lig_npro='PA45')
GROUP BY com_ncli
HAVING COUNT(*) >=2;
```

8.2.1.4 Créer des requêtes de type « jointure »

Dans une base de données relationnelle, une opération de jointure permet d'associer des données venant par exemple de plusieurs tables et d'afficher ces données dans le résultat.

1) Afficher NCLI, NOM et PRENOM des différents clients avec le numéro des commandes qu'ils ont passées en 2025.

```
SELECT cli_ncli, cli_nom, cli_prenom, com_ncom
FROM tbl_client_cli
INNER JOIN tbl_commande_com
ON tbl_client_cli.cli_ncli=tbl_commande_com.com_ncli
AND YEAR(com_date)=2025
ORDER BY cli_nom, cli_prenom, com_ncom;
```

2) Associer les clients aux produits qu'ils ont commandés.

```
SELECT DISTINCT cli_ncli, cli_nom, cli_prenom, pro_libelle
FROM tbl_client_cli
INNER JOIN tbl_commande_com
ON tbl_client_cli.cli_ncli=tbl_commande_com.com_ncli
INNER JOIN tbl_lignecom_lig
ON tbl_commande_com.com_ncom=tbl_lignecom_lig.lig_ncom
INNER JOIN tbl_produit_pro
ON tbl_lignecom_lig.lig_npro=tbl_produit_pro.pro_npro
ORDER BY cli_ncli, cli_nom, cli_prenom, pro_libelle;
```

3) Afficher NCLI, NOM et PRENOM des différents clients avec le numéro des commandes qu'ils ont passées en 2025. On veut que les clients qui n'ont pas commandé en 2025 soient affichés également (cas d'une jointure externe).

```
SELECT cli_ncli, cli_nom, cli_prenom, com_ncom
FROM tbl_client_cli
LEFT OUTER JOIN tbl_commande_com
ON tbl_client_cli.cli_ncli=tbl_commande_com.com_ncli
AND YEAR(com_date)=2025
ORDER BY cli_nom, cli_prenom, com_ncom;
```

8.2.2 Créer, modifier, utiliser et supprimer une vue

Pour rappel, une vue est une requête enregistrée dans la base de données sous la forme d'un objet. Elle permet de présenter les données issues d'une ou plusieurs tables comme si elles formaient une table unique, sans dupliquer les données.

- Syntaxe permettant de créer une vue :

```
CREATE VIEW v_client_produit AS
SELECT DISTINCT cli_ncli, cli_nom, cli_prenom, pro_libelle
FROM tbl_client_cli
INNER JOIN tbl_commande_com
ON tbl_client_cli.cli_ncli=tbl_commande_com.com_ncli
INNER JOIN tbl_lignecom_lig
ON tbl_commande_com.com_ncom=tbl_lignecom_lig.lig_ncom
INNER JOIN tbl_produit_pro
ON tbl_lignecom_lig.lig_npro=tbl_produit_pro.pro_npro;
```

- Syntaxe permettant de modifier une vue :

```
ALTER VIEW v_client_produit AS
SELECT DISTINCT cli_ncli, cli_nom, cli_prenom, pro_libelle, lig_prix
FROM tbl_client_cli
INNER JOIN tbl_commande_com
ON tbl_client_cli.cli_ncli=tbl_commande_com.com_ncli
INNER JOIN tbl_lignecom_lig
ON tbl_commande_com.com_ncom=tbl_lignecom_lig.lig_ncom
INNER JOIN tbl_produit_pro
ON tbl_lignecom_lig.lig_npro=tbl_produit_pro.pro_npro;
```

- Syntaxe permettant de faire appel à une vue :

```
SELECT *
FROM v_client_produit
ORDER BY cli_nom, cli_prenom;
```

- Syntaxe pour supprimer une vue :

```
DROP VIEW v_client_produit;
```