جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

كلية العلوم والتقنيات
FACULTÉ DES SCIENCES ET TECHNIQUES

Module (I241)

Base de Données

Pr. AZROUR MOURADE **Pr. ZEROUAL IMAD**

2019-2020

Table des matières

Chapitre I : Introduction Générale 4
I. Introduction 4
II. Quelques définitions 4
II.1. Base de données 4
II.2. Base de données informatisée 5
II.2. Système de Gestion des Base de données 5
III. Un peu d'histoire 6
IV. Objectifs des bases de données 7
V. Les différents types de bases de données 9
IV.1. Base de données hiérarchique 9
IV.2. Base de données réseau 9
IV.3. Base de données orientée texte 10
IV.4. Base de données SQL (relationnelle) 10
IV.5. Base de données distribuée 10
IV.6. Base de données cloud 11
IV.7. Base de données NoSQL 11
IV.8. Base de données orientée objets 11
IV.9. Base de données orientée graphe 11
Chapitre II : Le modèle conceptuel de données 12
I. Introduction 12
II. Éléments constitutifs du modèle 13
II.1. Entité 13
II.2. Attribut ou propriété, valeur 14
II.3. Identifiant ou clé 14
II.4. Association ou relation 14
II.5. Cardinalité 15
III. Compléments sur les associations 16
III.1. Associations plurielles 16
III.2. Association réflexive 16
IV. Règles pour une bonne modélisation 17
IV.1. Règles sur les noms 17
IV.2. Règles de fusion/suppression d'entités/associations 17

IV.3. Normalisation des type-entités et type-associations 18
V. Élaboration d'un modèle entités-associations 20
V.1. Étapes de conceptions d'un modèle entités-associations 20
V.2. Conseils divers 21
Chapitre III : Le Modèle logique de données (Relationnel) 23
I. Introduction 23
II. Règles de passage du MCD au MLD 23
II.1. Règle numéro 1 23
II.2. Règle numéro 2 24
II.3. Règle numéro 3 25
II.4 Cas particulier 25
II.5 L'héritage 26
III. L'algèbre relationnelle 27
III.1. Les opérations de l'algèbre relationnelle 28
III.2. Les opérateurs ensemblistes 28
Chapitre IV : Gestion des données avec le langage SQL 34
I. Introduction 34
II. Les commandes du langage SQL 34
II.1. DDL (Data Definition Language) 34
II.2. DQL (Data Query Language) 35
II.3. DML (Data Manipulation Language) 36
II.4. DCL (Data Control Language) 37
II.5. TCL (Transaction Control Language) 38
II.6. Fonctions SQL 39
Chapitre VI : Conclusion Générale 40

Avant-propos

Actuellement, les différentes organisations peuvent gérer leurs données d'une manière très efficace grâce aux systèmes de gestion de base de données, par suite développer et installer les applications utilisant ces données stockées. Les bases de données d'aujourd'hui joue un rôle très important au cœur du système d'information des organisations. Dans ce cours, nous intéressant aux bases de données relationnelles. Ce type des bases de données sont créées à l'aide de modèle relationnel, et en utilisant les principes de l'algèbre relationnelle.

Dans le premier chapitre, les concepts fondamentaux de base de données sont présentés. Plus précisément on a défini les mots clé qu'il faut savoir avant d'entrer dans la création d'une base de données, ensuit on a présenté une brève démarche sur les étapes essentielles pour la conception d'une telle base de données relationnelle. Le deuxième chapitre est totalement réservé au ***modèle conceptuel de données***, dans ce chapitre deux concepts principaux sont étudiés : entité et association. Le passage du modèle entités-associations ***au modèle relationnel*** et l'*algèbre relationnelle* est prévu dans le troisième chapitre. Le chapitre quatre est entièrement consacré au langage SQL (*Structured Query Language*) qui peut être considéré comme le langage d'accès normalisé aux bases de données relationnelles

Ce document est considéré comme le support pédagogique du cours « Base de Données» réservé aux étudiants du la faculté des sciences et techniques d'Errachidia, Université Moulay Ismail, Royaume du Maroc.

Chapitre I : Introduction Générale

I. Introduction

En 1960, les développeurs informatiques ont inventé ce qu'on appelé aujourd'hui les bases de données (BD), afin de faire face aux problèmes liés aux systèmes des fichiers et simplifier la gestion qualitative et quantitative des données numériques. Les applications informatiques permettent la création et la gestion des bases de données sont dites les **S**ystèmes de **G**estion des **B**ases de **D**onnées **R**elationnelles (SGBDR). Tous ces systèmes sont basés sur un même langage pour concevoir les bases de données d'une manière identique indépendamment de système utilisé, c'est le langage SQL (**S**tructured **Q**uery **L**anguage).

Ainsi, dans une activité d'une organisation, une grande quantité d'informations est utilisé afin d'assurer le bon déroulement de cette organisation. Afin d'être disponibles pour la prise des décisions, ces informations doivent être sauvegardé, mis-à-jour, accessible et traitées. En plus, l'information est un outil qui permet à une organisation de communiquer avec son environnement. Par conséquent, les administrateurs doivent avoir une habilité qui les aide à faire le traitement et la gestion de l'information et la rendre utilisable et lucrative pour leur organisation. Cette habilité ne peut être possible que par la maitrise des outils de gestion des données comme SGBDR.

Dans ce chapitre, nous allons étudier les concepts de base en relation avec les bases de données. Plus précisément on va voir les définitions fondamentales, l'objectif et les avantages des bases de données, les types des bases de données et vers la fin en va répondre à la question pourquoi les bases de données ?

II. Quelques définitions

II.1. Base de données

On désigne par le terme **base de données** un ensemble structuré et organisé qui permet de stocker une grande quantité d'informations afin de les exploiter (ajout, mise à jour, recherche, suppression).

II.2. Base de données informatisée

Une base de données informatisée est définie comme un ensemble structuré de données sauvegardées sur des supports de stockages qui sont accessibles par des systèmes informatiques. Ces données peuvent être par suite interrogées et mises à jour par les utilisateurs de système.

Exemple :

Les données dans une base de données relationnelle sont organisées sous forme des tableaux. Dans ce qui suit ce tableau représente la liste des employés d'une entreprise.

Matricule	*Nom*	*Prénom*	*Grade*	*Salaire*
200	Azizi	Mohammed	Cadre	10000
201	Alaoui	Karim	Employé	5500
202	Boudlal	Noura	Assistant	3500
203	Hasnaoui	Said	Employé	5000
....	....	....	....	...

II.2. Système de Gestion des Base de données

Un Système de Gestion des Base de données (**SGBD**) est un logiciel qui est responsable de structurer, stocker, mettre à jour et maintenir d'une base de données. Il offre une interface entre les gestionnaires et les utilisateurs d'une part et les données d'autre part.

Exemple :

Il existe plusieurs systèmes de gestion des bases de données, on site dans les lignes suivants quelques exemples:

- **Oracle Database** : est un système de gestion de base de données relationnelle (SGBDR) qui depuis l'introduction du support du modèle objet dans sa version 8 peut être aussi qualifiée de système de gestion de base de données relationnel-objet (SGBDRO).

- **PostgreSQL** est un système de gestion de base de données relationnelle et objet (SGBDRO). C'est un outil libre disponible selon les termes d'une licence de type BSD.
- **Microsoft Access** est composé de plusieurs programmes : le moteur de base de données Microsoft Jet, un éditeur graphique, une interface de pour interroger les bases de données, et le langage de programmation Visual Basic for Applications.
- **MongoDb** est un système de gestion de base de données orienté documents, répartissable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données.
- **MySQL** est un système de gestion de bases de données relationnelles (SGBDR). Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde.
- **Microsoft SQL Server** est un système de gestion de base de données (SGBD) en langage SQL incorporant entre autres un SGBDR (SGBD relationnel) développé et commercialisé par la société Microsoft. Il fonctionne sous les OS Windows et Linux (depuis mars 2016).
- **OpenOffice.org Base** est le système de gestion de bases de données relationnelles (SGBDR) proposé par les suites bureautiques libres Libre Office et Apache OpenOffice.

III. Un peu d'histoire

- En 1970, E.F. CODD invente le modèle relationnel qui est basé sur une algèbre relationnelle.
- En 1977, apparition du langage SEQUEL (Structured English Query Language) et mise en place du Système R pour implémenté SEQUEL. SEQUEL a été développé pour devenir vers la fin SQL (Structured Query Language).
- En 1981, la société ORACLE développe la première version de son système de gestion de base de données relationnelle (SGBDR).
- En 1982, IBM lance SQL/DS pour son environnement et l'ANSI (American National Standard Institute) lance un projet de normalisation d'un langage relationnel.
- En 1983, IBM lance DB2 pour l'environnement MVS.

- En 1986, la société SYBASE lance son SGBDR conçu suivant le modèle Client-Serveur.
- En 1989, La première norme SQL (SQL-1) de l'ISO (International Standard Organisation) apparaît.
- La norme SQL est passée à SQL-2, puis SQL-3, SQL-2003, SQL-2011...

IV. Objectifs des bases de données

Généralement, les bases de données sont développées pour atteindre les objectifs suivants :

- ***Indépendance physique***

  L'indépendance physique signifie que la manière par laquelle les données sont déterminées doit être indépendante des structures de stockages usés. Il permet donc de réaliser l'indépendance des structures de stockage aux structures de données du monde réel, c'est-à-dire entre le schéma interne et le schéma conceptuel. On pourra par exemple ajouter un index, regrouper deux fichiers en un, changer l'ordre ou le codage des données dans un article, sans mettre en cause les entités et associations définies au niveau conceptuel.

- ***Indépendance logique***

  L'indépendance logique doit permettre :

  - ✓ D'avoir des vues différentes des mêmes données par des utilisateurs différents.
  - ✓ L'évolution d'un schéma externe sans remettre en cause les autres schémas externes.
  - ✓ Modifier le schéma logique sans avoir des effets au niveau des applications

- ***Gestion des données par un langage non procédural***

  Généralement, il existe deux types des utilisateurs qui peuvent manœuvrer les bases de données : les utilisateurs interactifs et les programmeurs. Le premier type des utilisateurs peuvent consulter, faire les mis à jour, ou bien supprimer les données. Ils ne pas forcement des informaticiens, cela leur nécessite d'avoir des langages simples. Ces derniers doivent interactifs et supportés par un SGBD, depuis les langages de commandes semi-formels jusqu'aux langages graphiques.

- ***Administration centralisée des données***

  Pour bien contrôler les données d'une manière efficace, de résoudre les conflits entre divers points de vue et d'optimiser les accès aux données et en utilisant un outil informatique, les créateurs des bases de données ont conçu à maitre toutes ces activités dans les mains des responsables nommées les administrateurs de données.

- ***Non-redondance des données***

  Durant les premières années d'apparition de l'informatique ; les applications ont chacune ses propres données. De ce fait, les données dupliquées ont constitué un problème majeur conduit généralement à de nombreuses duplications de données, à savoir la nécessité des grandes zone mémoire pour enregistrées les même données plusieurs fois les mêmes données. Cependant, après l'apparition des bases de données, les données dupliquées sont intégrées en un seul fichier réparti entre plusieurs applications. L'administration centralisée des données conduisait donc naturellement à la non-duplication physique des données afin d'éviter les mises à jour multiples.

- ***Partage des données***

  Les données sauvegardées dans une base de données doivent être partagées entre les applications dans le temps. Chaque application doit avoir le droit d'accéder aux données comme si elle est seule à les utiliser, sans attendre les autres applications. Toutefois, les données peuvent être modifiées par une autre application simultanément.

- ***Sécurité des données***

  Les données d'une base de données doivent être sécurisées de deux manières. Premièrement, elles doivent être sécurisées contre les accès non autorisés ou contre les malveillants. Pour atteindre cet objectif, on doit offrir un mécanisme pour gérer (autoriser, modifier ou enlever) les droits d'accès de tous les utilisateurs de système qui gère la base de données. Deuxièmement, la sécurité des données doit aussi être satisfaite dans les cas où le système est en panne. Le système gérant la base de données doit être capable de restaurer les données endommagées ou perdues après la panne.

- ***Cohérence des données***

  Bien que les redondances anarchiques entre données soient évitées par l'objectif précédent, les données vues par l'utilisateur ne sont pas indépendantes. Au niveau d'ensemble de données, il peut exister certaines dépendances entre données. Par exemple, une donnée représentant le nombre de commandes d'un client doit correspondre au nombre de commandes dans la base. Plus simplement, une donnée élémentaire doit respecter un format et ne peut souvent prendre une valeur quelconque.

- ***Accès aux données d'une manière efficace***

  Les performances en termes de débit (nombre de transactions types exécutées par seconde) et de temps de réponse (temps d'attente moyen pour une requête type) sont un problème clé des SGBD. L'objectif de débit élevé nécessite un ***overhead (plafond)*** minimal dans la gestion des tâches accomplie par le système. L'objectif de bons temps de réponse implique qu'une requête courte d'un utilisateur n'attende pas une requête longue d'un autre utilisateur. Il faut donc partager les ressources (unités centrales, unités d'entrées-sorties) entre les utilisateurs en optimisant l'utilisation globale et en évitant les pertes en commutation de contextes.

V. Les différents types de bases de données

Dans le cadre d'une analyse de données, il est nécessaire que les données en provenance de plusieurs fichiers puissent être liées. C'est pourquoi **différents types de bases de données ont été développé pour répondre à ces exigences** :

IV.1. Base de données hiérarchique

Les bases de données hiérarchiques sont parmi **les plus anciennes bases de données**. Au sein de ce concept, les enregistrements sont organisés dans une structure d'arborescence. Chaque niveau d'enregistrements découle sur un ensemble de catégories plus petites.

IV.2. Base de données réseau

Les bases de données réseau sont également parmi les plus anciennes. Plutôt que de proposer des liens uniques entre différents ensembles de données à divers niveaux, les bases de données réseaux créent **des liens multiples entre les**

ensembles en plaçant des liens, ou des pointeurs, sur un ensemble d'enregistrements ou un autre. La vitesse et la polyvalence des bases de données réseau ont conduit à une adoption massive de ce type de base données au sein des entreprises ou dans le domaine du e-commerce.

IV.3. Base de données orientée texte

Une base données orientée texte se présente sous la forme d'un **fichier (une table) au format .txt ou .ini**. Un fichier est un fichier texte, ou un fichier combinant du texte avec un fichier binaire. En général, dans ces bases de données, chaque ligne ne comporte qu'un enregistrement. La plupart des bases de données pour PC sont des bases de données orientées texte.

IV.4. Base de données SQL (relationnelle)

Les bases de données relationnelles ont été inventées en 1970 par E.F. Codd de IBM. Il s'agit de documents tabulaires dans laquelle **les données sont définies afin d'être accessibles** et de pouvoir être réorganisées de différentes manières.

Les bases de données relationnelles sont constituées d'un ensemble de tableaux. Au sein de ces tableaux, les **données sont classées par catégorie**. Chaque tableau comporte au moins une colonne correspondant à une catégorie. Chaque colonne comporte un certain nombre de données correspondant à cette catégorie.

Le langage standard pour les bases de données relationnelles est **le Structured Query Language (SQL)**. Les bases de données relationnelles sont facilement extensibles, et de nouvelles catégories de données peuvent être ajoutées après la création de la base de données originale sans avoir besoin de modifier toutes les applications existantes.

IV.5. Base de données distribuée

Une Base de données distribuée est une base de données dont **certaines portions sont stockées à plusieurs endroits** physiques. Le traitement est réparti ou répliqué entre différents points d'un réseau.

Les bases de données distribuées **peuvent être homogènes ou hétérogènes**. Dans le cas d'un système de base de données distribuée homogène, tous les emplacements physiques fonctionnent avec le même hardware et tournent sous le même système d'exploitation et les mêmes applications de bases de données. Au

contraire, dans le cas d'une base de données distribuée hétérogène, le hardware, les systèmes d'exploitation et les applications de bases de données peuvent varier entre les différents endroits physiques.

IV.6. Base de données cloud

Dans ce cadre, elle est **optimisée ou directement créée pour les environnements virtualisés**. Il peut s'agir d'un cloud privé, d'un cloud public ou d'un cloud hybride.

Les **bases de données cloud offrent plusieurs avantages** comme la possibilité de payer pour la capacité de stockage et la bande passante en fonction de l'usage. Par ailleurs, il est possible de changer l'échelle sur demande. Ces bases de données offrent aussi une disponibilité plus élevée.

IV.7. Base de données NoSQL

Les bases de données NoSQL sont utiles **pour les larges ensembles de données distribuées**. En effet, les bases de données relationnelles ne sont pas conçues pour le Big Data, et les ensembles de données trop larges peuvent poser des problèmes de performances.

IV.8. Base de données orientée objets

Les **objets créés à l'aide de langage de programmation orientés objets** sont généralement stockés sur des bases de données relationnelles. Toutefois, en réalité, les bases de données orientées objets sont plus adaptées pour stocker ce type de contenu.

IV.9. Base de données orientée graphe

Une base de données orientée graphe est un **type de Database NoSQL utilisant la théorie des graphes** pour stocker, cartographier et effectuer des requêtes sur les relations entre les données. Les bases de données graphe sont constituées de nœuds et de bords. Chaque nœud représente une entité, et chaque bord représente une connexion entre les nœuds. Les bases de données graphes gagnent en popularité dans le domaine des analyses d'interconnexions. Par exemple, les entreprises peuvent utiliser une BD graphe pour **miner des données sur ses clients à partir des réseaux sociaux**.

Chapitre II : Le modèle conceptuel de données

I. Introduction

La modélisation est l'étape initiatrice du processus de conception d'une base de données. Elle permet d'abstraire le problème réel pour en faire une reformulation qui trouvera une solution dans le cadre technologique d'un SGBD. Une ou plusieurs modélisations intermédiaires sont donc utiles, le modèle entités-associations constitue l'une des premières et des plus courantes. Ce modèle est appelé : **MERISE**.

MERISE (*Méthode d'Étude et de Réalisation Informatique pour les Systèmes d'Entreprise*) est un langage de spécification le plus répandu dans la communauté de l'informatique des systèmes d'information, et notamment dans les bases de données. Une représentation MERISE permet de valider des choix par rapport aux objectifs, de quantifier les solutions retenues, de mettre en œuvre des techniques d'optimisation et enfin de guider jusqu'à l'implémentation. Ce modèle est composé de trois niveaux pour la représentation des données qui sont :

- **Niveau conceptuel** : Le modèle conceptuel des données (**MCD**) décrit les entités du monde réel, en termes d'objets, de propriétés et de relations, indépendamment de toute technique d'organisation et d'implantation des données. Ce modèle se concrétise par un schéma entités-associations représentant la structure du système d'information, du point de vue des données.
- **Niveau organisationnel** : Le Modèle Organisationnel des Données (**MOD**) a comme mission d'intégrer dans l'analyse les critères liés à l'organisation étudiée. Le MOD fera préciser les notions de temporalité, de chronologie des opérations, d'unité de lieu, définira les postes de travail, l'accès aux bases de données…
- **Niveau logique** : Le modèle logique des données (**MLD**) précise le modèle conceptuel par des choix organisationnels. Il s'agit d'une transcription (également appelée dérivation) du MCD dans un formalisme adapté à une implémentation ultérieure, au niveau physique, sous forme de base de données relationnelles ou réseau, ou autres
- **Niveau physique** : Le modèle physique des données (**MPD**) permet d'établir la manière concrète dont le système sera mis en place (SGBD retenu).

Dans ce chapitre nous allons étudier le modèle conceptuel de données.

II. Éléments constitutifs du modèle

Le Modèle Conceptuel des Données (MCD) est l'élément le plus connu de MERISE et certainement le plus utile. Il permet d'établir une représentation claire des données du Système d'Information et définit les dépendances fonctionnelles de ces données entre elles.

Les éléments de base utilisés pour la représentation d'un MCD sont les suivants :

- ✓ L'objet ou entité,
- ✓ L'association,
- ✓ La propriété.

L'objet est une entité ayant une existence propre. L'association est un lien ou relation entre objets sans existence propre. La propriété est la plus petite donnée d'information décrivant un objet ou une association.

II.1. Entité

Une entité : *est un objet, une chose concrète ou abstraite qui peut être reconnue distinctement et qui est caractérisée par son unicité.*

Exemple : Ahmed Sefrioui, le Rouge et le Noir roman de Balzac, Mercredis, etc.

Un type-entité : *désigne un ensemble d'entités qui possèdent une sémantique et des propriétés communes.*

Exemple : Les personnes, les livres et les voitures sont des type-entité.

En effet, dans le cas d'un livre par exemple, les *propriétés* comme le titre, l'auteur et date de publication, ne changent pas de nature.

Une entité est habituellement appelée occurrence ou instance de son type-entité. La représentation graphique de type-entité (*Livre*) sans ses propriétés associées, est faite en utilisant un rectangle comme dans cette figure suivante :

Les type-entité Personne, caractérisé par un nom, un prénom et une date de naissance, et Voiture, caractérisé par un nom et une puissance fiscale, ne peuvent pas être regroupés car ils ne partagent leurs propriétés (le prénom est une chaîne de caractères et la puissance fiscale un nombre). Les type-entité Personne, caractérisé par un nom et un prénom, et Livre, caractérisé par un titre et un auteur, possèdent tous les deux attributs du type chaîne de caractères. Pourtant, ces deux type-entités ne peuvent

pas être regroupés car ils ne partagent pas une même sémantique : le nom d'une personne n'a rien à voir avec le titre d'un livre, le prénom d'une personne n'a rien à voir avec un auteur.

II.2. Attribut ou propriété, valeur

Un attribut (ou une propriété) : est une caractéristique associée à un type-entité ou à un type-association.

Exemples : le nom d'une personne, le titre d'une livre, la puissance d'une voiture.

Au niveau du type-entité ou du type-association, chaque attribut possède un domaine qui définit l'ensemble des valeurs possibles qui peuvent être choisies pour lui (entier, chaîne de caractères, booléen, . . .). Au niveau de l'entité, chaque attribut possède une valeur compatible avec son domaine.

II.3. Identifiant ou clé

Un identifiant (ou clé) : est constitué par un ou plusieurs de ses attributs qui doivent avoir une valeur unique pour chaque entité ou association de ce type.

Il est donc impossible que les attributs constituant l'identifiant d'un type-entité (respectivement type association) prennent la même valeur pour deux entités (respectivement deux associations) distinctes. Exemples d'identifiant : le numéro de la carte nationale pour une personne, le numéro d'immatriculation pour une voiture, le code ISBN d'un livre pour un livre.

La représentation graphique d'un exemple de type-entité comportant quatre attributs dont un est un identifiant : deux livres peuvent avoir le même titre, le même prix et sont écrit par le même auteur, mais ils ne peuvent pas avoir le même code ISBN.

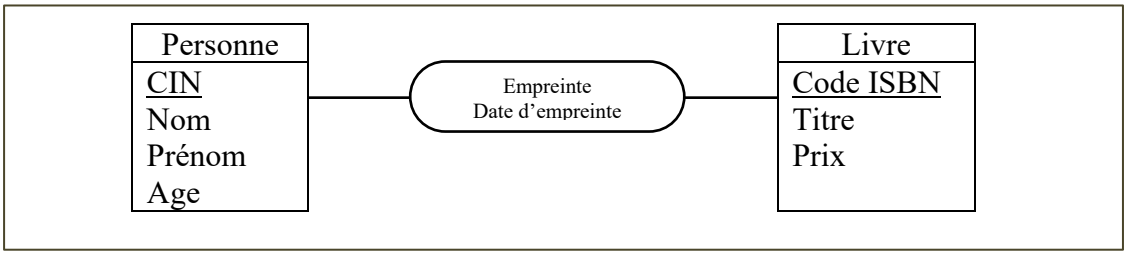
Livre
Code ISBN
Titre
Prix
Auteur

II.4. Association ou relation

Définition d'association : Une association (ou une relation) est un lien entre plusieurs entités.

Définition de type-association : Un type-association (ou un type-relation) désigne un ensemble de relations qui possèdent les mêmes caractéristiques. Le type-association décrit un lien entre plusieurs type-entités.

La représentation graphique d'un exemple de type-association liant deux type-entités se fait comme suit :



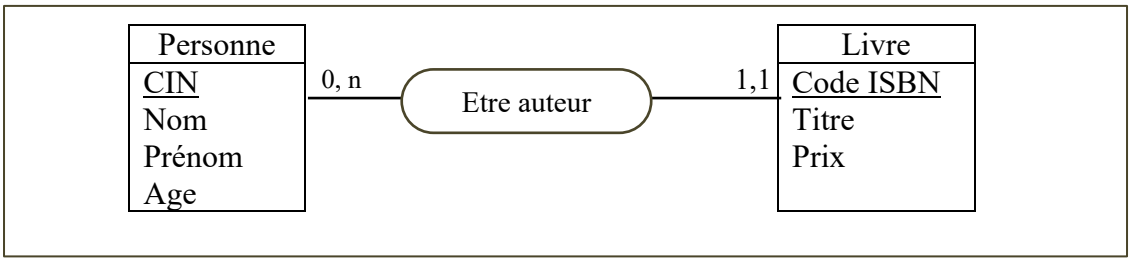
II.5. Cardinalité

Définition -cardinalité- La cardinalité d'une patte reliant un type-association et un type-entité précise le nombre de fois minimal et maximal d'interventions d'une entité du type-entité dans une association du type association. La cardinalité minimale doit être inférieure ou égale à la cardinalité maximale.

L'expression de la cardinalité est obligatoire pour chaque patte d'un type-association. Une cardinalité minimale est toujours ***0*** ou ***1*** et une cardinalité maximale est toujours ***1*** ou ***n***.

Exemple :

Dans cet exemple pédagogique, on suppose qu'un livre ne peut posséder qu'un auteur. Une personne peut être l'auteur de 0 à n livre, mais un livre ne peut être écrit que par une personne.



Les cardinalités admises sont :

0,1 : une occurrence du type-entité peut exister tout en étant impliquée dans aucune association et peut être impliquée dans au maximum une association.

0,n : c'est la cardinalité la plus ouverte ; une occurrence du type-entité peut exister tout en étant impliquée dans aucune association et peut être impliquée, sans limitation, dans plusieurs associations.

1,1 : une occurrence du type-entité ne peut exister que si elle est impliquée dans exactement (au moins et au plus) une association.

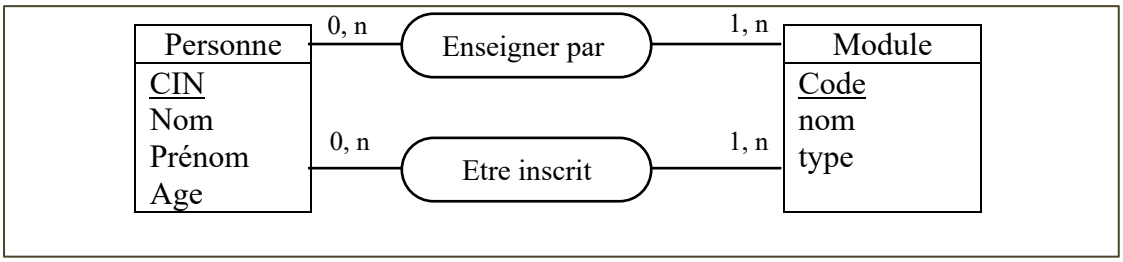
1,n : une occurrence du type-entité ne peut exister que si elle est impliquée dans au moins une association.

III. Compléments sur les associations

III.1. Associations plurielles

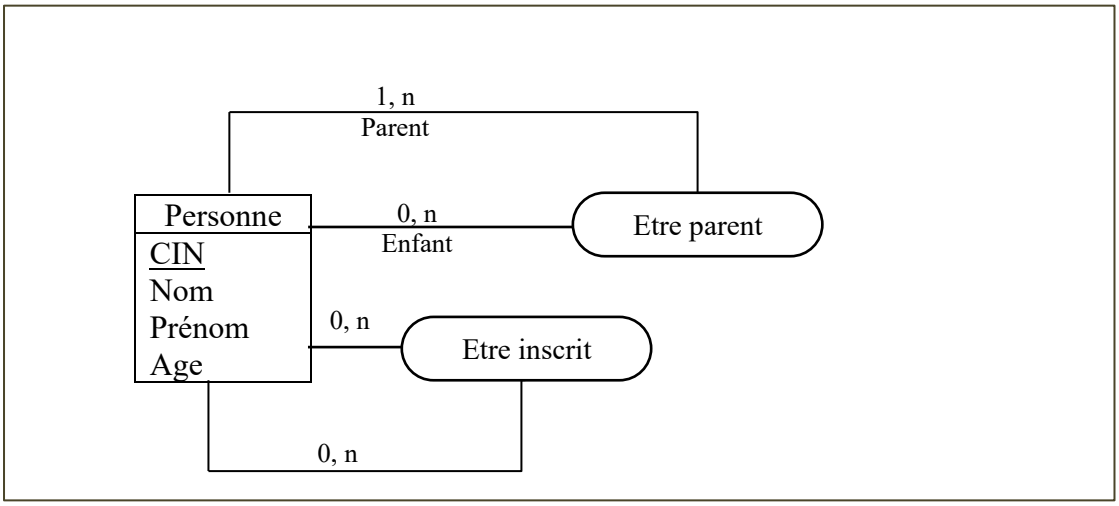
Dans ce cas deux mêmes entités peuvent être associées plusieurs fois. L'exemple suivant montre que l'entité personne est associe à l'entité module deux fois.

Un module est enseigné par 1 ou n personnes (professeurs). Et au même temps 1 ou n personnes (étudiants) sont inscrit dans un module.



III.2. Association réflexive

Un type-association est qualifié de réflexif quand il matérialise une relation entre un type-entité et lui-même. Dans l'exemple suivant, l'entité personne à une association réflexible sur elle-même.

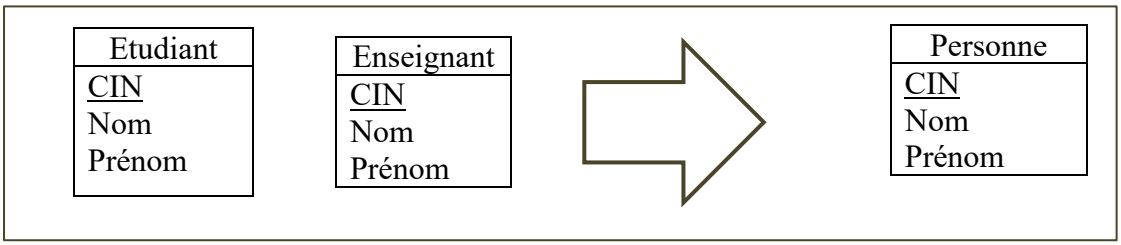


IV. Règles pour une bonne modélisation

IV.1. Règles sur les noms

- Dans un modèle entités-associations, le nom d'un type-entité, d'un type-association ou d'un attribut doit être unique.

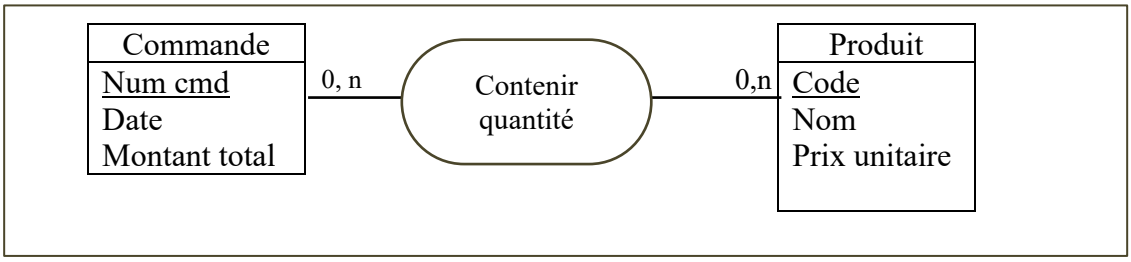
Exemple :



- Eviter d'ajouter un attribut dérivé d'autres attributs, que ces autres attributs se trouvent dans le même type-entité ou pas.

Exemple :

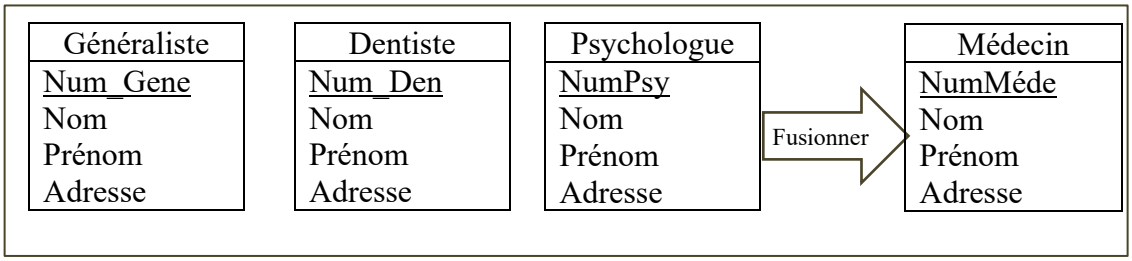
L'erreur commit dans cet exemple est l'ajout de champ « Montant total » qui peut être déduit à partir de « Prix unitaire » x « quantité ».

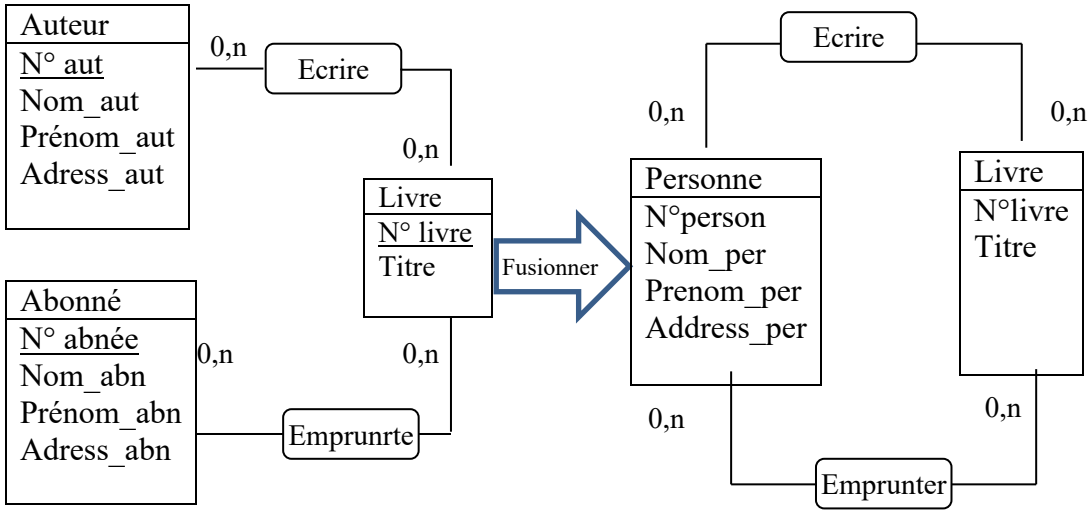


IV.2. Règles de fusion/suppression d'entités/associations

Fusion

Il faut factoriser les type-entités quand c'est possible.





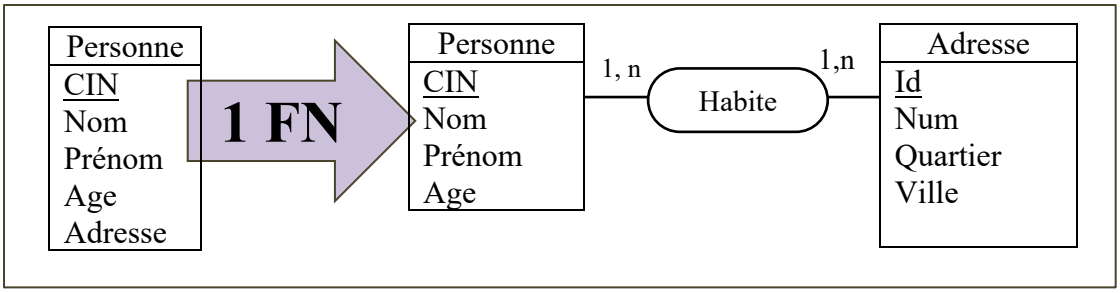
IV.3. Normalisation des type-entités et type-associations

Les formes normales sont différents stades de qualité qui permettent d'éviter la redondance, source d'anomalies. La normalisation peut être aussi bien effectuée sur un modèle entités-associations, où elle s'applique sur les type-entités et type-associations, que sur un modèle relationnel. Il existe 5 formes normales principales et deux extensions. Plus le niveau de normalisation est élevé, plus le modèle est libère de redondances. Un type-entité ou un type-association en forme normale de niveau n est automatiquement en forme normale de niveau n−1.

Première forme normale (1FN)

Un type-entité ou un type-association est en première forme normale si tous ses attributs sont élémentaires, c'est-à-dire non décomposables.

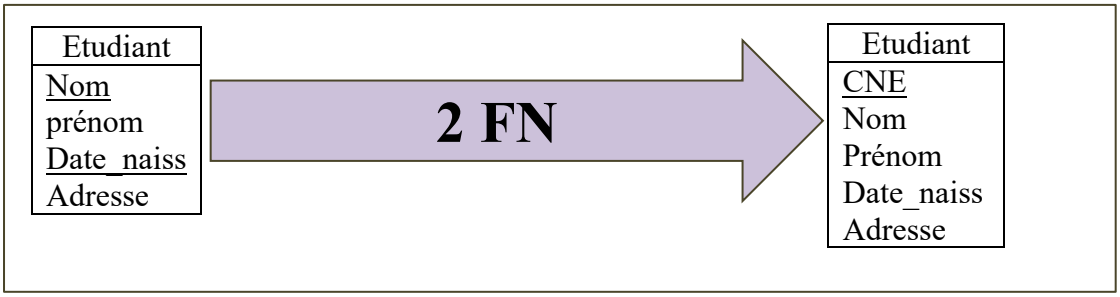
Un attribut composite doit être décomposé en attributs élémentaires ou faire l'objet d'une entité supplémentaire. L'élémentarité d'un attribut est toutefois fonction des choix de gestion. Par exemple, la propriété Adresse peut être considérée comme élémentaire si la gestion de ces adresses est globale. Par contre, s'il faut pouvoir considérer les codes postaux, les noms de rues..., il convient d'éclater la propriété Adresse en Adresse (au sens numéro d'appartement, numéro et nom de rue...), Code postal et Ville. En cas de doute, il est préférable (car plus général) d'éclater une propriété que d'effectuer un regroupement.



Deuxième forme normale (2FN)

Un type-entité ou un type-association est en deuxième forme normale si, et seulement si, il est en première forme normale et si tout attribut n'appartenant pas à la clé dépend de la totalité de cette clé.

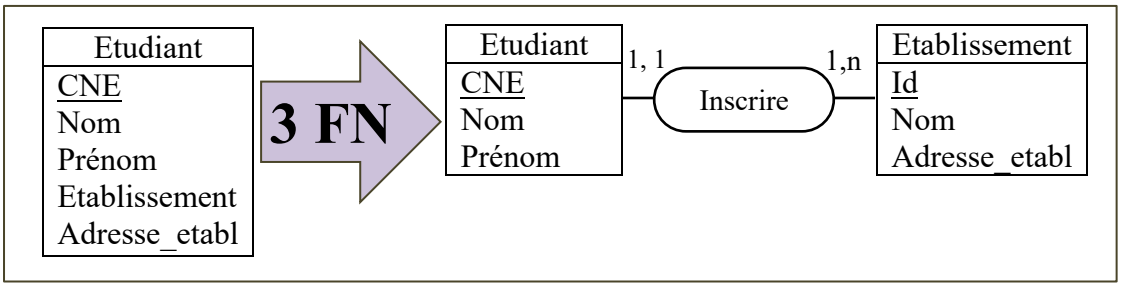
Autrement dit, les attributs doivent dépendre de l'ensemble des attributs participant à la clé. Ainsi, si la clé est réduite à un seul attribut, ou si elle contient tous les attributs, le type-entité ou le type-association est, par définition, forcément en deuxième forme normale.



Troisième forme normale (3FN)

Un type-entité ou un type-association est en troisième forme normale si, et seulement si, il est en deuxième forme normale et si tous ses attributs dépendent directement de sa clé et pas d'autres attributs.

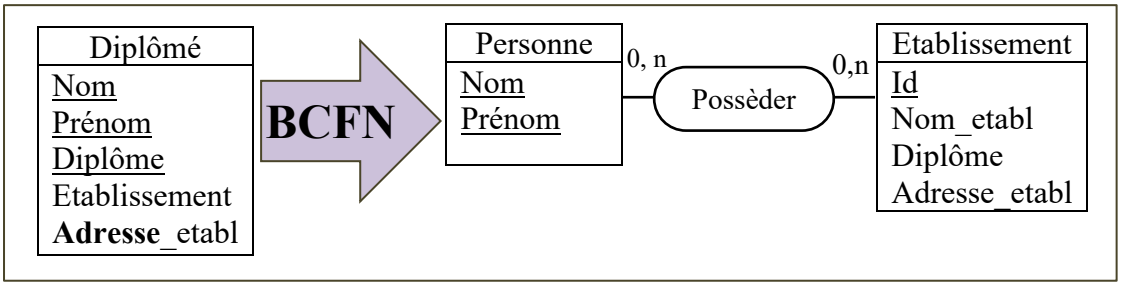
Dans l'exemple suivant ***« adresse_établ*** » dépend de « ***établissement*** » et pas de code national d'étudiant « ***CNE*** ». Donc, il faut la normaliser selon la forme normale 3.



Forme normale de Boyce-Codd (BCFN)

Un type-entité ou un type-association est en forme normale de Boyce-Codd si, et seulement si, il est en troisième forme normale et si aucun attribut faisant partie de la clé dépend d'un attribut ne faisant pas partie de la clé.

Dans l'exemple ci-après, le diplômé est une personne identifiée par le nom, le prénom et le diplôme. Alors que le diplôme qui est une partie de la clé dépend de l'établissement qui n'est pas une partie de la clé. Alors, il faut la normaliser selon la forme normal Boyce-Codd (BCNF).



V. Élaboration d'un modèle entités-associations

V.1. Étapes de conceptions d'un modèle entités-associations

Pour concevoir un modèle entités-associations, vous devrez certainement passer par une succession d'étapes. Nous les décrivons ci-dessous dans l'ordre chronologique. Sachez cependant que la conception d'un modèle entités-associations est un travail non linéaire. Vous devrez régulièrement revenir à une étape précédente et vous n'avez pas besoin d'en avoir terminé avec une étape pour commencer l'étape suivante**.**

Recueil des besoins – C'est une étape primordiale. Inventoriez l'ensemble des données à partir des documents de l'entreprise, d'un éventuel cahier des charges et plus généralement de tous les supports de l'information. N'hésitez pas à poser des questions.

Tri de l'information – Faites le tri dans les données recueillies. Il faut faire attention, à ce niveau, aux problèmes de synonymie/polysémie. En effet, les attributs ne doivent pas être redondants. Par exemple, si dans le langage de l'entreprise on peut parler indifféremment de *référence d'article* ou de *n° de produit* pour désigner la même chose, cette caractéristique ne devra se concrétiser que par un unique attribut dans le modèle. Inversement, on peut parler d'adresse pour désigner l'adresse du fournisseur et l'adresse du client, le contexte permettant de lever l'ambiguïté. Par contre, dans le

modèle, il faudra veiller à bien distinguer ces deux caractéristiques par deux attributs distincts.

Identification des type-entités – Le repérage d'attributs pouvant servir d'identifiant permet souvent de repérer un type-entité. Les attributs de ce type-entité sont alors les attributs qui dépendent des attributs pouvant servir d'identifiant. Attention, un même concept du monde réel peut être représenté dans certains cas comme un attribut et dans d'autres cas comme un type-entité, selon qu'il a ou non une existence propre. Par exemple, la marque d'une automobile peut être vue comme un attribut du type-entité *Véhicule* de la base de données d'une préfecture mais aussi comme un type-entité *Constructeur automobile* dans la base de données du Ministère de l'Industrie. Lorsqu'on ne parvient pas à trouver d'identifiant pour un type-entité, il faut se demander s'il ne s'agit pas en fait d'un type-association. Si ce n'est pas le cas, un identifiant arbitraire numérique entier peut faire l'affaire.

Identification des type-associations – Identifiez les type-associations reliant les type-entités du modèle. Le cas échéant, leur affecter les attributs correspondants. Il est parfois difficile de faire un choix entre un type-entité et un type-association. Par exemple, un mariage peut être considéré comme un type-association entre deux personnes ou comme un type-entité pour lequel on veut conserver un numéro, une date, un lieu, . . ., et que l'on souhaite manipuler en tant que tel.

Vérification du modèle – Vérifiez que le modèle respecte bien les règles que nous avons énoncées et les définitions concernant la normalisation des type-entités et des type-associations. Le cas échéant, opérez les modifications nécessaires pour que le modèle soit bien formé.

V.2. Conseils divers

Concernant le choix des noms Pour les type-entités, choisissez un nom commun décrivant le type-entité (ex : Étudiant, Enseignant, Matière). Certain préfèrent mettre le nom au pluriel (ex : Étudiants, Enseignants, Matières). Restez cependant cohérents, soit tous les noms de type-entité sont au pluriel, soit ils sont tous au singulier.

Pour les type-association, choisissez un verbe à l'infinitif, éventuellement à la forme passive ou accompagné d'un adverbe (ex : Enseigner, Avoir lieu dans). Pour les attributs, utilisez un nom commun au singulier éventuellement accompagné du nom du type-entité ou du type-association dans lequel il se trouve (ex : nom de client, numéro d'article).

Concernant le choix des identifiants des type-entités Évitez les identifiants composés de plusieurs attributs (comme, par exemple, un identifiant formé par les attributs *nom* et prénom d'un type-association Personne) car : – ils dégradent les performances du SGBD, – mais surtout l'unicité supposée par une telle démarche finit généralement, tôt ou tard, par être démentie ! Évitez les identifiants susceptibles de changer au cours du temps (comme la plaque d'immatriculation d'un véhicule). Évitez les identifiants du type chaîne de caractère. En fait, il est souvent préférable de choisir un identifiant arbitraire de type entier pour les types entités. Cet identifiant deviendra une clé primaire dans le schéma relationnel et le SGBD l'incrémentera automatiquement lors de la création de nouvelles instances. L'inconvénient de cette pratique est qu'il devient possible de se retrouver avec deux instances du type-entités représentant le même objet mais avec deux numéros différents. Malgré cet inconvénient, cette politique de l'identifiant reste largement avantageuse dans la pratique et permet, en outre, de s'affranchir (en la satisfaisant automatiquement) de la deuxième forme normale.

Bien distinguer les concepts de données et de traitements La modélisation conceptuelle de données exclut la représentation des traitements futurs sur ces données. Toutefois, elle nécessite la connaissance de ces traitements pour prévoir les données élémentaires indispensables à ceux-ci. En conséquence, il existe une confusion fréquente entre les concepts de données et de traitements. Par exemple, la facturation est un traitement qui nécessite de connaître toutes les caractéristiques d'une commande. Par contre, la facturation ne se traduit ni par un type-entité, ni par un type-association dans le schéma entités-associations.

Chapitre III : Le Modèle logique de données (Relationnel)

I. Introduction

L'étape Modèle Logique de Données (**MLD**) se situe chronologiquement juste après l'étape MCD et revient à :

- Transformer toute entité en table;
- Chaque identifiant est considéré comme clé primaire;
- Représenter les valeurs prises par les cardinalités maximum de chaque association soit :
    - Par l'ajout d'une clé étrangère dans une table existante;
    - Par la création d'une nouvelle table dont la clé primaire est obtenue par concaténation de clés étrangères correspondant aux entités liées.

On emploie souvent l'abréviation **MLD** et quelquefois, les abréviations suivantes sont employées :

- **MLDR** : Modèle Logique de Données Relationnelles.
- **MRD** : Modèle Relationnel de Données.
- **MLRD** : Modèle Relationnel Logique de Données.

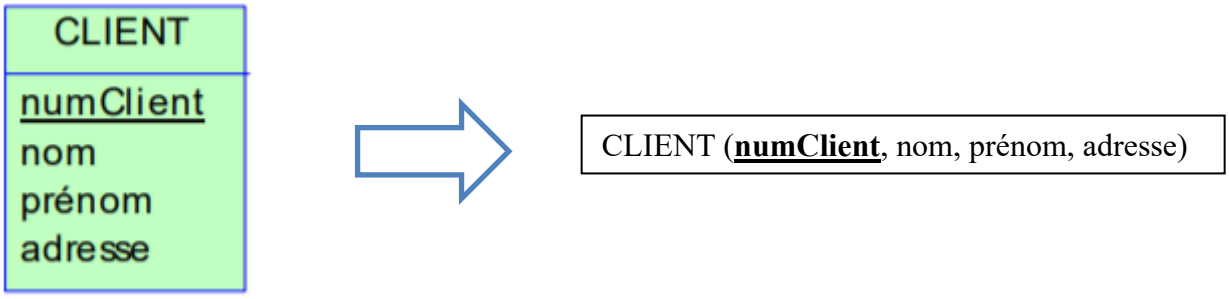
II. Règles de passage du MCD au MLD

II.1. Règle numéro 1

La première règle du passage du MCD au MLD est résumé dans les étapes suivantes :

- Une entité du MCD devient une table/relation.
- Son identifiant devient la clé primaire de la table.
- Les autres propriétés deviennent les attributs de la table.

Exemple :



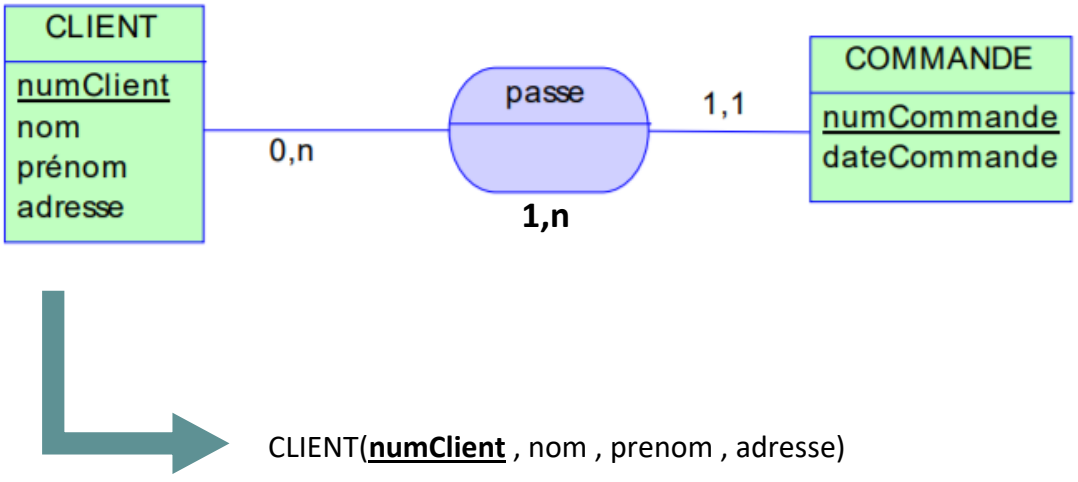
numClient est une clé primaire de la table CLIENT.

II.2. Règle numéro 2

La deuxième règle du passage du MCD au MLD s'applique lorsqu'on a une association de type **1:N** :

- L'association donc se traduit par la création d'une clé étrangère dans la relation correspondante à l'entité côté « 1 ».
- Cette clé étrangère référence la clé primaire de la relation correspondant à l'autre entité.

Exemple :



CLIENT(**numClient** , nom , prenom , adresse)

numClient : clé primaire de la table CLIENT

COMMANDE(**numCommande** ,dateCommande , **#numClient**)

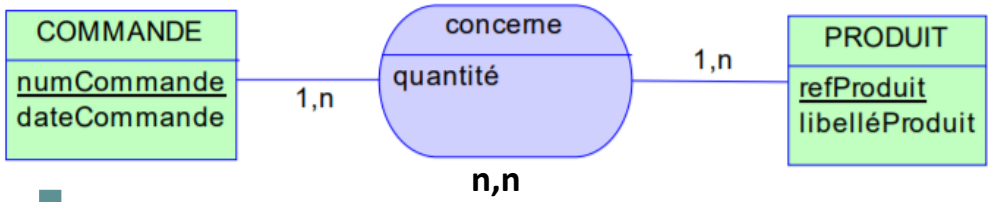
numCommande : clé primaire de la table COMMANDE
#numClient : clé étrangère qui référence numClient de la table CLIENT

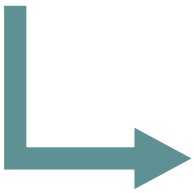
II.3. Règle numéro 3

La troisième règle du passage du MCD au MLD s'applique lorsqu'on a une association de type **N:N** :

- L'association se traduit par la création d'une table dont la clé primaire est composée des clés étrangères référençant les relations correspondant aux entités liées par l'association.
- Les éventuelles propriétés de l'association deviennent des attributs de la relation.

Exemple :





COMMANDE(**numCommande** ,dateCommande)

PRODUIT(**refProduit**, libelleProduit)

CONCERNE(**#numCommande** , **#refProduit** , quantité)

Remarque :

Si le nom du MCD n'est pas **significatif**, on peut renommer le nom de la table, par exemple :

- CONCERNE(**#numCommande , #refProduit ,** quantité**)**

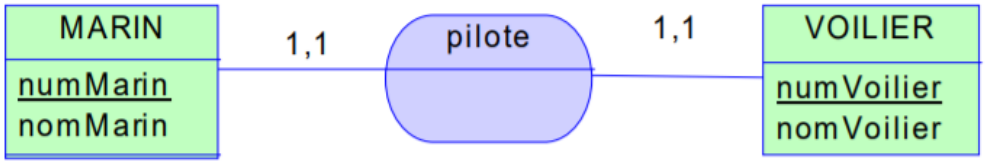
devient :

- ✓ LIGNE_DE_COMMANDE(**#numCommande , #refProduit ,** quantité**)**

II.4 Cas particulier

Ce cas s'agit de l'association de type **1:1,** Cela dépend fonctionnellement sur l'entité la plus important.

Exemple :



1. Si fonctionnellement, le marin est le plus important, on ajoute les attribués de l'entité VLOILIER à celles de l'entité MARIN sans considérer la clé de VOILIER comme clé étranger :

   MARIN(**numMarin** , nomMarin , numVoilier , nomVoilier)

2. Si fonctionnellement, le voilier est le plus important, on ajoute les attribués de l'entité MARIN à celles de l'entité VOILIER sans considérer la clé de MARIN comme clé étranger :

   VOILIER(**numVoilier** , nomVoilier , numMarin , nomMarin)

3. Si le modèle peut évoluer ou si on a une distinction fonctionnelle forte entre marin et voilier, on a deux possibilités :

   A. VOILIER(numVoilier , nomVoilier , #numMarin)

      MARIN(numMarin , nomMarin)

      OU

   B. VOILIER(**numVoilier** , nomVoilier)

      MARIN(**numMarin** , nomMarin, # **numVoilier**)

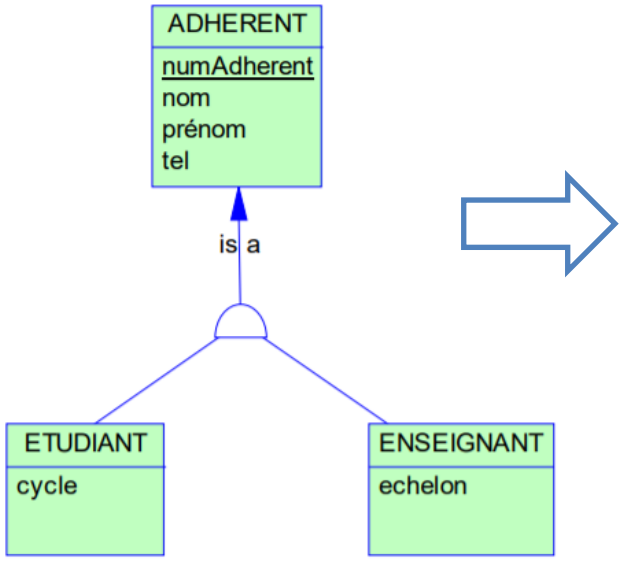
II.5 L'héritage

L'héritage est l'association entre deux entités permettant d'exprimer que l'une est plus générale que l'autre. Dire qu'entité A' hérite de A équivaut à dire que A' est une fille de A. On peut également dire que A est une généralisation de A' ou que A est une mère de A'. Dans le cas d'héritage, on suit les étapes suivantes :

- L'entité mère et les entités filles se transforment en table.
- L'identifiant de l'entité mère devient la clé primaire de la table qui correspond à l'entité mère et aux tables qui correspondent aux entités filles.
- Les propriétés des entités se transforment en attributs des tables.

- Les clés primaires des « tables filles » sont aussi des clés étrangères qui référencent la clé primaire de la « table mère.«
- Un champ est ajouté dans la « table mère » pour permettre de typer les occurrences, c'est-à-dire d'identifier quelle est la « table fille » concernée.

Exemple :



ADHERENT (**numAdherent** , nom, prénom, tel, **type**)

ETUDIANT (**numAdherent** , cycle)

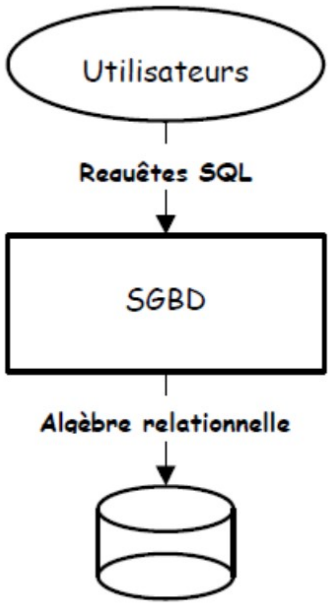
ENSEIGNANT (**numAdherent** , echelon)

III. L'algèbre relationnelle

L'algèbre relationnelle est le langage de manipulation qu'utilise le SGBD pour effectuer des opérations sur les relations (tables). L'objectif de ce langage est de localiser des données dans la base qui répondent à certains critères.

Remarque :

Les requêtes SQL soumises par l'utilisateur sont traduites par le SGBD en opérations de l'algèbre relationnelle.



III.1. Les opérations de l'algèbre relationnelle

L'algèbre relationnelle possède huit opérateurs : certains opérateurs sont **ensemblistes** (selon la théorie des ensembles), d'autres sont **relationnels** (spécifiques à l'algèbre relationnelle).

Opérateur relationnel	Opérateurs ensemblistes
Sélection	**Produit cartésien**
Projection	**Union**
Jointure	**Intersection**
Division	**Différence**

Remarque :

Toutes les opérations à deux opérandes sauf que la **Sélection** et la **Projection** qui nécessitent un seul opérande.

III.2. Les opérateurs ensemblistes

Les opérateurs ensemblistes sont les mêmes qu'en mathématiques, dans la théorie des ensembles.

Union ∪ :

L'union de deux tables est l'ensemble des occurrences qui appartiennent soit à la première table, soit à la deuxième, soit aux deux tables. C'est la traduction du OU logique.

Formalisme :

R = R1 U R2 ou R = UNION (R1 , R2)

Exemple :

R :

PIECE	FOURNISS
écrou	pierre
écrou	paul
boulon	alice

S :

PIECE	FOURNISS
écrou	pierre
boulon	pierre
boulon	alice

RUS :

PIECE	FOURNISS
écrou	pierre
écrou	paul
boulon	alice
boulon	pierre

Intersection ∩ :

L'intersection de deux relations est l'ensemble des occurrences qui sont présentes dans les deux relations. C'est la traduction du ET logique.

Formalisme :

R = R1 ∩ R2 ou R = INTERSECTION (R1, R2)

Exemple :

R :

PIECE	FOURNISS
écrou	pierre
écrou	paul
boulon	alice

S :

PIECE	FOURNISS
écrou	pierre
boulon	pierre
boulon	alice

R∩S :

PIECE	FOURNISS
écrou	pierre
boulon	alice

Différence – :

La différence entre deux tables est l'ensemble des occurrences qui appartiennent à une table sans appartenir à la seconde. **Attention**, cette opération a un sens. L'opération différence est non commutative.

Formalisme :

R = R1 - R2 ou R = DIFFERENCE (R1, R2)

Exemple :

R :

PIECE	FOURNISS
écrou	pierre
écrou	paul
boulon	alice

S :

PIECE	FOURNISS
écrou	pierre
boulon	pierre
boulon	alice

S - R :

PIECE	FOURNISS
boulon	pierre

Produit cartésien x :

Le produit cartésien de 2 tables consiste à combiner toutes les possibilités d'associations d'occurrences des 2 tables. Chaque ligne de R1 sera concaténée à chaque ligne de R2.

Formalisme :

R = R1 * R2 ou R = PRODUIT (R1, R2)

Exemple :

Elève	Num	Nom	Adresse	Age
	1	Bélaïd	Maisel	20
	2	Millot	CROUS	20
	3	Meunier	Maisel	21

UV	Code	Nbh	Coord
	IO	45	Lalevée
	BD	15	Carpentier

Elève X UV	Num	Nom	Adresse	Age	Code	Nbh	Coord
	1	Bélaïd	Maisel	20	IO	45	Lalevée
	2	Millot	CROUS	20	IO	45	Lalevée
	3	Meunier	Maisel	21	IO	45	Lalevée
	1	Bélaïd	Maisel	20	BD	15	Carpentier
	2	Millot	CROUS	20	BD	15	Carpentier
	3	Meunier	Maisel	21	BD	15	Carpentier

Restriction (Sélection) σ :

La sélection consiste à extraire d'une relation les occurrences (lignes) satisfaisant au(x) critère(s) de sélection.

Formalisme :

R2 = $\sigma_{\text{critère(s)}}$ (R1) ou SELECTION (R1, critère(s))

Critères de sélection :

- Opérateurs de comparaison :

  <, <=, =, >, >=, ? (entre un champ et une valeur)

- Opérateurs logiques :

  ET, OU (entre deux comparaison)

- NON (pour renverser la comparaison)

Exemple :

no-film	titre	durée	production	code-catégorie
10	Camille Claudel	150	Gaumont	COMD
20	Fenêtre sur cour	120	Pathé	COMD
25	Sueurs froides	115	Pathé	COMD
50	Cendrillon	140	UGC	DESA
64	Super Mondet II	10	Universal	DOCU
65	La vie des coccinelles	60.	UGC	DOCU
71	La guerre des étoiles I	120	Paramounth	COMD
...	...	...		...

Si on aimerait avoir les films de durée supérieure à 115, on écrit :

$\sigma_{\text{durée} > 115}$ (FILM) **ou** SELECTION (FILM, durée> 115)

Projection Π :

La projection d'une relation consiste en la mise en place d'une nouvelle relation en ne retenant que certaines colonnes (attributs) et en supprimant les occurrences en double.

Formalisme :

R2 = $\Pi_{\text{colonne 1, colonne 2, ...}}$ (R1) ou PROJECTION (R1, colonne 1, colonne 2, …)

Avec :

R2 est la table résultat, **R1** est la table utilisée par la projection. Aussi, il n'y a pas de duplication des occurrences.

Exemple :

no-film	titre	durée	production	code-catégorie
10	Camille Claudel	150	Gaumont	COMD
20	Fenêtre sur cour	120	Pathé	COMD
25	Sueurs froides	115	Pathé	COMD
50	Cendrillon	140	UGC	DESA
64	Super Mondet II	10	Universal	DOCU
65	La vie des coccinelles	60.	UGC	DOCU
71	La guerre des étoiles I	120	Paramounth	COMD
...	...	...		...

Si on aimerait avoir les titres et productions de tous les films, on écrit :

$\Pi_{\text{titre, production}}$ (FILM)

Ou

Projection (FILM, titre, production)

Division / :

La division permet de trouver les occurrences d'une table qui sont associées à toutes les occurrences d'une autre table (qui le plus souvent est le résultat d'une sélection). L'opération Division est non commutative.

Formalisme :

R = DIVISION (R1, R2) ou R = R1/ R2

Exemple :

Participant

Athlète	Epreuve
Daoud	200 m
Dorra	400 m
Daoud	400 m
Mahrez	110 m H
Daoud	110 m H
Mahrez	200 m

Epreuve

Epreuve
200 m
400 m
110 m H

Participant / Epreuve

Athlète
Daoud

Jointure ►◄ :

La jointure consiste à créer une nouvelle table à partir de deux tables ayant un champ commun (attribut) et vérifiant un critère de jointure.

Formalisme :

R = JOINTURE R1, R2 (R1.attr_jointure op_comparaison R2.attr_jointure)

Ou

R = R1 ►◄ R2 (R1.attr_jointure op_comparaison R2.attr_jointure)

Exemple :

Cours

Nom_cours	professeur	coefficient
Gestion de projet	Mme Gatri	2
Bases de données	Mme Mastouri	4
Analyse	Mr Dridi	2
Data Minig	Mme Mastouri	3

Notes

Etudiant	Nom cours	note
Ahmed	Gestion de projet	12
Amel	Bases de données	14
Imed	Bases de données	12
Zeineb	Gestion de projet	13

La jointure entre les deux tables Cours et Notes se fait à travers l'attribut **Nom_cours :**

- ✓ **Cours ►◄ Notes (Cours.Nom_cours = Notes.Nom_cours)**

Ou

- ✓ **JOINTURE** $_{\text{Cours, Notes}}$ **(Cours.Nom_cours = Notes.Nom_cours)**

Tableau de résultat :

Nom_cours	professeur	coefficient	Etudiant	note
Gestion de projet	Mme Gatri	2	Zeineb	13
Gestion de projet	Mme Gatri	2	Ahmed	12
Bases de données	Mme Mastouri	4	Amel	14
Bases de données	Mme Mastouri	4	Imed	12

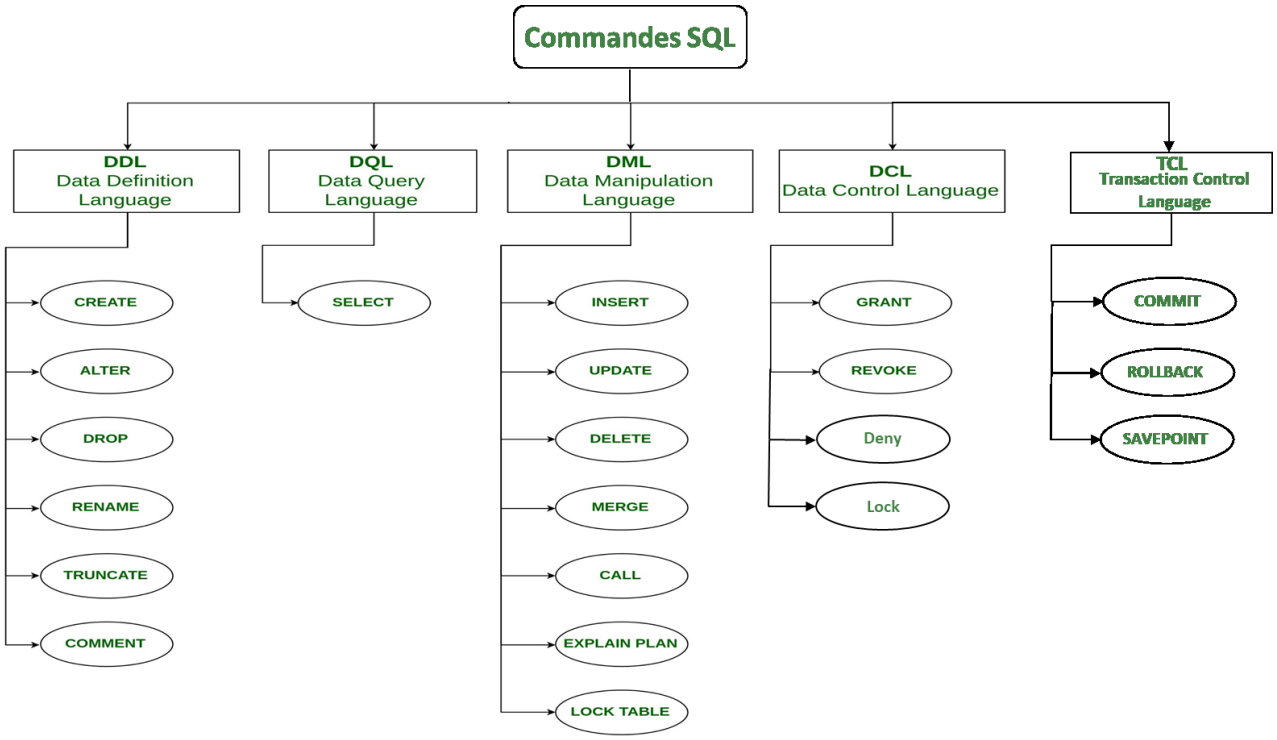
Chapitre IV : Gestion des données avec le langage SQL

I. Introduction

Le SQL (Structured Query Language), en français Langage de Requête Structurée, est un langage de requête structurée destiné pour gérer les bases de données relationnelles. Ce langage informatique normalisé est généralement employé par les développeurs et les programmeurs des applications informatiques afin de gérer leurs données.

II. Les commandes du langage SQL

Les commandes du langage SQL sont principalement classées en **cinq** catégories : DDL, DQL, DML, DCL, et TCL.



II.1. DDL (Data Definition Language)

Les DDL sont les commandes SQL utilisées pour manipuler les **structures** de données d'une base de données, et non les données elles-mêmes.

On distingue typiquement **six** types de commandes SQL de définition de données :

- **CREATE** : création d'une structure de données ;
- **ALTER** : modification d'une structure de données ;
- **DROP** : suppression d'une structure de données ;
- **RENAME** : renommage d'une structure de données ;
- **TRUNCATE** : suppression toutes les données d'une structure de données sans supprimer la structure elle-même ;
- **COMMENT** : définition ou modification d'un commentaire associé à une structure de données.

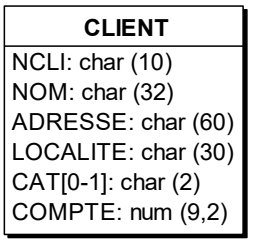
Ces commandes peuvent principalement porter sur les structures de données de type suivantes :

- **TABLE** : table ;
- **INDEX** : indice ;
- **VIEW** : Vue (table virtuelle) ;
- **SEQUENCE** : suite de nombres ;
- **SYNONYM** : synonyme ;
- **USER** : utilisateur.

Exemple :

Création d'une table et de ses colonnes :

```
CREATE TABLE CLIENT (NCLI     char(10),
                     NOM      char(32),
                     ADRESSE  char(60),
                     LOCALITE char(30),
                     CAT      char(2),
                     COMPTE   decimal(9,2) );
```



II.2. DQL (Data Query Language)

Les DQL sont les commandes SQL utilisées pour **récupérer** les données de la base de données. Une telle commande SQL est l'instruction **SELECT** :

- **SELECT** : sélection et récupération de données dans une structure de données (ex : Table).

La requête **SELECT** peut posséder presque toutes les commandes possibles :

- SELECT *****
- **FROM** table
- **WHERE** condition
- **GROUP BY** expression

- **HAVING** condition
- **{ UNION | INTERSECT | EXCEPT }**
- **ORDER BY** expression
- **LIMIT** count
- **OFFSET** start

Exemple :

Soit la table Université suivante :

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Etienne	21000

Le résultat de la commande suivante est :

SLECT nomU, effectif **FROM** Universite

nomU	effectif
INSA	36000
UCB	15000
UJF	10000
UJM	21000

II.3. DML (Data Manipulation Language)

Les DML sont les commandes SQL utilisées pour manipuler les **données** d'une base de données. Cette catégorie inclut la plupart des commandes SQL, par exemple :

- **INSERT** : insertion de données dans une table ;
- **DELETE** : suppression de données d'une table ;
- **UPDATE** : mise à jour de données d'une table ;
- **MERGE** : sélection de la table à modifier ou pour insérer des nouvelles données ;
- **OPTIMIZE** : réorganisation du stockage physique des données et reconstruit les index afin d'améliorer les performances dans certains cas ;
- **CASE** … **WHEN** : utilisation des conditions de type "si / sinon" pour retourner un résultat disponible entre plusieurs possibilités. Le CASE peut être utilisé dans n'importe quelle instruction ou clause, telle que SELECT, UPDATE, DELETE, WHERE, ORDER BY ou HAVING ;
- **EXPLAIN** : affichage du plan d'exécution d'une requête SQL. Cela permet de savoir de quelle manière le SGBD va exécuter la requête et s'il va utiliser des index et lesquels. L'instruction EXPLAIN est à utiliser juste avant un SELECT et elle ne renverra pas les résultats du SELECT mais plutôt une analyse de cette requête.

Exemple :

Soit la table **CLIENT** suivant :

id	prenom	nom	ville	age
1	Rébecca	Armand	Saint-Didier-des-Bois	24
2	Aimée	Hebert	Marigny-le-Châtel	36
3	Marielle	Ribeiro	Maillères	27

Apres l'exécution de la commande INSERT suivante :

INSERT INTO CLIENT (prenom, nom, ville, age)

VALUES ('Hilaire', 'Savary', 'Conie-Molitard', 58);

La table CLIENT devient :

id	prenom	nom	ville	age
1	Rébecca	Armand	Saint-Didier-des-Bois	24
2	Aimée	Hebert	Marigny-le-Châtel	36
3	Marielle	Ribeiro	Maillères	27
4	Hilaire	Savary	Conie-Molitard	58

A noter :

Lorsque le champ à remplir est de type VARCHAR ou TEXT (le cas des champs : prénom, nom, et ville) il faut indiquer le texte entre guillemet simple. En revanche, lorsque la colonne est un numérique tel que INT (le cas de champ âge) il n'y a pas besoin d'utiliser de guillemet, il suffit juste d'indiquer le nombre.

II.4. DCL (Data Control Language)

Les DCL sont les commandes SQL utilisées pour **contrôler l'accès** aux données d'une base de données. On distingue typiquement **quatre** types de commandes SQL de contrôle de données :

- **GRANT** : autorisation d'un utilisateur à effectuer une action ;
- **DENY** : interdiction à un utilisateur d'effectuer une action ;
- **REVOKE** : annulation d'une commande de contrôle de données précédente ;
- **LOCK** : verrouillage sur une structure de données.

Exemple :

La commande suivante accorde une autorisation **SELECT** sur la table « **employés** » à un utilisateur de base données nommé « **user1** ».

GRANT SELECT **ON** employés **TO** user1;

II.5. TCL (Transaction Control Language)

Les TCL sont les commandes SQL utilisées pour gérer les **transactions** dans la base de données. Ceux-ci sont utilisés pour gérer les modifications apportées aux données dans un tableau par des instructions DML. Il permet également de regrouper les instructions en transactions logiques.

On distingue typiquement **trois** types de commandes SQL de contrôle de données :

- **COMMIT** : validation d'une transaction en cours et l'enregistrer de façon permanente ;
- **ROLLBACK** : annulation d'une transaction en cours ;
- **SAVEPOINT** : enregistrement de façon temporairement une transaction afin que vous puissiez revenir à ce point chaque fois que nécessaire.

Exemple :

Soit la table **CLIENT** suivant :

id	prenom	nom	ville	age
1	Rébecca	Armand	Saint-Didier-des-Bois	24
2	Aimée	Hebert	Marigny-le-Châtel	36
3	Marielle	Ribeiro	Maillères	27
4	Hilaire	Savary	Conie-Molitard	58

On va supprimer tout enregistrement qui a comme prénom « Hilaire » avec la commande (DML) DELETE :

DELETE from CLIENT **where** prenom = 'Hilaire';

La table CLIENT devient:

id	prenom	nom	ville	age
1	Rébecca	Armand	Saint-Didier-des-Bois	24
2	Aimée	Hebert	Marigny-le-Châtel	36
3	Marielle	Ribeiro	Maillères	27

Pour annuler l'effet de cette transaction (DELETE) et retourner à l'état précèdent de la Table CLIENT, il suffit alors d'utiliser la commande ROLLBACK comme suit :

DELETE from CLIENT **where** prenom = 'Hilaire';

ROLLBACK ;

II.6. Fonctions SQL

Les fonctions SQL permettent d'effectuer des requêtes plus élaborées, par exemple :

- **Fonctions d'agrégation :**
  - **SUM()** : calculer la somme d'un set de résultat ;
  - **MAX()** : obtenir le résultat maximum (fonctionne bien pour un entier) ;
  - **MIN()** : obtenir le résultat minimum ;
  - **COUNT()** : compter le nombre de lignes dans un résultat ;
  - **AVG()** : calculer la moyenne sur un ensemble d'enregistrements de type numérique.
- **Fonctions mathématiques / numérique**
  - **ROUND()** : arrondir la valeur et retourner soit un nombre entier, ou de choisir le nombre de chiffre après la virgule ;
  - **RAND()** : sélectionner un nombre aléatoire à virgule, compris entre 0 et 1.
- **Fonctions de chaînes de caractères :**
  - **UPPER()** : afficher une chaîne en majuscule ;
  - **LOWER()** : afficher une chaîne en minuscule ;
  - **CONCAT()** : concaténer des chaînes de caractères ;
  - **REVERSE()** : retourner une chaîne de caractère en inversant l'ordre des caractères ;
  - **REPLACE()** : remplacer des caractères alphanumérique dans une chaîne de caractère ;
  - **LENGTH()** : calculer la longueur d'une chaîne de caractères ;
  - **SUBSTRING()/SUBSTR()** : segmenter une chaîne de caractère ;
  - **TRIM()** : supprimer des caractères au début et en fin d'une chaîne de caractère.
- **Fonctions de dates et d'heures :**
  - **NOW()** : date et heure actuelle ;
  - **DATE_FORMAT()** : formater une donnée DATE dans le format indiqué ;
  - **MONTH()** : extraire le numéro de mois à partir d'une date au format AAAA-MM-JJ ;
  - **DATEDIFF()** : déterminer l'intervalle entre 2 dates spécifiées ;
- **Fonctions de chiffrements :**
  - **MD5()** : chiffrer une chaîne de caractère en un entier hexadécimal de 32 caractères. La fonction MD5() utilise l'algorithme **RSA**.

Chapitre VI : Conclusion Générale

Les bases de données ont pris une place importante en informatique, et particulièrement dans le domaine de la gestion. L'étude des bases de données à conduit au développement de concepts, méthodes et algorithmes spécifiques, notamment pour gérer les données en support de stockage (exemple : disques durs). En effet, dès le début de la découverte de l'informatique, les informaticiens ont constaté que la taille de la mémoire vive n'est pas capable de charger toutes les données d'une base de données. Aujourd'hui, cette supposition est aussi approuvable car le masse des données augments toujours surtout après l'apparition de Web2.0 et les objets connectés. De ce fait, les bases de données futures devront être dans la mesure de gérer des grandes masses de données, distribuées géographiquement, et accessibles par des milliers d'utilisateurs.

Références

1) Merise - Guide pratique (3e édition), septembre 2018, ISBN : 9782409015342.
2) https://sql.sh/
3) http://tecfaetu.unige.ch/staf/staf-h/tassini/staf2x/Heidi/last_bd.htm