Les Bases de données

Filière : Achats et Supply Chain Management

Sommaire

I. Introduction aux bases de données

1. Notion de base de données

Il est difficile de donner une définition exacte de la notion de base de données. Une définition très générale pourrait être :

Définition 1 :

Base de données : *Un ensemble organisé d'informations avec un objectif commun.*

Peu importe le support utilisé pour rassembler et stocker les données (papier, fichiers, etc.), dès lors que des données sont rassemblées et stockées d'une manière organisée dans un but spécifique, on parle de base de données.

Plus précisément, on appelle base de données un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour, recherche de données). Bien entendu, dans le cadre de ce cours, nous nous intéressons aux bases de données informatisées.

Définition 2 :

Base de données informatisée : *Une base de données informatisée est un ensemble structuré de données enregistrées sur des supports accessibles par l'ordinateur (disque dur, cd-rom, clé USB, ...), représentant des informations du monde réel et pouvant être interrogées et mises à jour par plusieurs utilisateurs.*

Avantages d'utilisation des bases de données :

- **stocker** de **gros volumes** d'informations
- **partager** des informations par une communauté de personnes
- **gérer l'accès** à ces informations
- gérer des **informations cohérentes et non-redondantes**

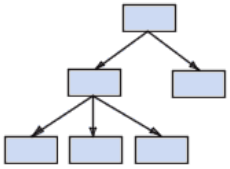
Une fois la base de données spécifiée, on peut y insérer des données, les récupérer, les modifier et les détruire. C'est ce qu'on appelle manipuler les données. Les données peuvent être manipulées non seulement par un Langage spécifique de Manipulation des Données (LMD), mais aussi par des langages de programmation classiques.

2. Modèles de base de données

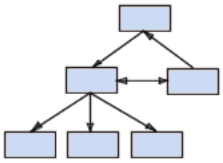
Un modèle de données est un ensemble de concepts et de règles de composition de ces concepts permettant de décrire des données.

Il existe principalement quatre types de modèles :

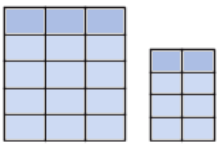
- **Le modèle hiérarchique** : les données sont classées hiérarchiquement, selon une arborescence descendante. Ce modèle utilise des pointeurs entre les différents enregistrements. Il s'agit du premier modèle de BD



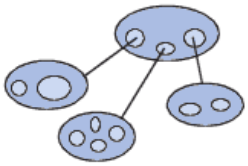
- **Le modèle réseau** : comme le modèle hiérarchique ce modèle utilise des pointeurs vers des enregistrements. Toutefois la structure n'est plus forcément arborescente dans le sens descendant.



- **Le modèle relationnel** : les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes). La manipulation de ces données se fait selon la théorie mathématique des relations.



- **Le modèle objet** : les données sont stockées sous forme d'objets, c'est-à-dire de structures appelées classes présentant des données membres. Les champs sont des instances de ces classes.



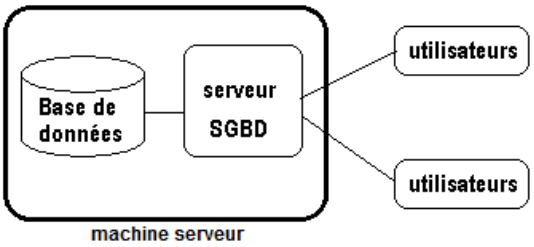
A la fin des années 90 les bases relationnelles sont les bases de données les plus répandues (environ trois quarts des bases de données).

3. Système de gestion de base de données (SGBD)

Un SGBD est un ensemble de logiciels chargés d'assurer les fonctions minimales suivantes :

- Le maintien de la cohérence des données entre elles,
- Le contrôle d'intégrité des données accédées,
- Les autorisations d'accès aux données,
- Les opérations classiques sur les données (consultation, insertion, modification, suppression)

Actuellement, la plupart des SGBD fonctionnent selon un mode client/serveur. Le serveur (sous-entendu la machine qui stocke les données) reçoit des requêtes de plusieurs clients et ceci de manière concurrente. Le serveur analyse la requête, la traite et retourne le résultat au client.



Quelques SGBD connus et utilisés :

- MS Access
- Microsoft SQL
- MySQL
- PostgreSQL
- Oracle
- IBM DB2
- Sybase
- Informix

II. Modèle relationnel

1. Introduction :

Pour concevoir une base de données une modélisation est au préalable nécessaire, après avoir analysé l'ensemble des informations à stocker et les utilisations que l'on voudra en faire. Cette modélisation va faire des liens entre les données, les contrôler (sécurité, validité), les traiter à l'aide d'un programme informatique, les rendre accessibles à un grand nombre d'utilisateurs, gérer des volumes de données de très grande taille. C'est pour cela qu'une BDR (base de données relationnelle) contient la plupart du temps plusieurs tables qui sont reliées entre elles par des associations.

2. La méthode Merise :

MERISE (Méthode d'Étude et de Réalisation Informatique pour les Systèmes d'Entreprise) est une méthode française de conception et de développement de systèmes d'information créée dans les années 1970-1980. Elle reste largement utilisée dans les pays francophones.

Objectifs de Merise :

- Analyser et concevoir des systèmes d'information
- Séparer les données des traitements
- Garantir la qualité et la pérennité des systèmes
- Faciliter la communication entre utilisateurs et informaticiens

3. Les Principes Fondamentaux de MERISE

3.1. Les Trois Cycles

MERISE structure un projet selon trois cycles :

Cycle d'Abstraction (3 niveaux)

1. **Niveau Conceptuel** : QUOI ? (Indépendant de l'organisation et de la technique)
2. **Niveau Organisationnel (Logique)** : QUI ? QUAND ? OÙ ? (Organisation du travail)
3. **Niveau Opérationnel (Physique)** : COMMENT ? (Choix techniques et matériels)

Cycle de Vie (phases du projet)

1. Schéma directeur
2. Étude préalable
3. Étude détaillée
4. Réalisation
5. Mise en œuvre
6. Maintenance

Cycle de Décision

- Validation à chaque étape
- Retours possibles aux étapes précédentes

3.2. Séparation Données/Traitements

MERISE distingue clairement :

- **L'aspect DONNÉES** : ce que le système mémorise
- **L'aspect TRAITEMENTS** : ce que le système fait

Cette séparation génère 6 modèles différents (3 niveaux × 2 aspects).

4. Les Modèles de Données

La méthode Merise est une méthode française de conception de systèmes d'information. Le **MCD (Modèle Conceptuel de Données)** est l'un de ses piliers, permettant de représenter de manière abstraite la structure des données d'un système d'information.

4.1. Les Concepts Fondamentaux du MCD

4.1.1. Méthodologie de Conception d'un MCD

Étape 1 : Identification des Entités

- Analyser le cahier des charges
- Repérer les objets pertinents du domaine
- Nommer les entités au singulier

Étape 2 : Identification des Propriétés (attributs)

- Pour chaque entité, lister ses caractéristiques
- Identifier l'identifiant unique (clé primaire)
- Éliminer les propriétés calculées

Étape 3 : Identification des Associations

- Repérer les liens entre entités
- Nommer les associations (verbe ou expression)
- Identifier les propriétés portées par les associations

Étape 4 : Détermination des Cardinalités

- Appliquer les règles de gestion
- Poser les bonnes questions : "Combien de fois minimum/maximum ?"

Étape 5 : Validation et Normalisation

- Vérifier la cohérence du modèle
- Éliminer les redondances
- Valider avec les utilisateurs

4.1.2. Etape 1 : Identifier les Entités

Une **entité** représente un objet du monde réel ayant une existence propre et pertinente pour le système d'information.

Elles peuvent être par exemple :

- un acteur : client, fournisseur, auteur
- un « objet » : produit, document, message
- un flux : livraison, commande, transport

Caractéristiques :

- Représentée par un rectangle
- Possède un nom au singulier (ex: Client, Produit, Commande)
- Contient des propriétés (attributs)

Exemples :

Un client est caractérisé par un numéro, un nom, un prénom, une adresse et un numéro de téléphone.

Client
NumClient
Nom
Prénom
Adresse
Téléphone

4.1.3. Etape 2 : Les Propriétés (Attributs)

Les **propriétés** décrivent les caractéristiques d'une entité ou d'une association.

Types de propriétés :

- **Identifiant**: propriété qui identifie de manière unique chaque occurrence de l'entité (souligné)
- **Propriétés simples** : attributs descriptifs de l'entité

Exemple :

Dans l'entité Client, le **NumClient** est l'identifiant.

4.1.4. Etape 3 : Les Associations (Relations)

Une **association** représente un lien sémantique entre deux ou plusieurs entités.

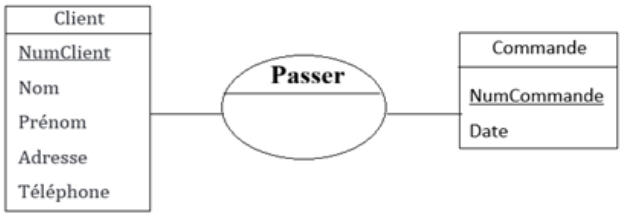
- Une association permet de mettre en relation une ou plusieurs entités
- Son existence est conditionnée par les entités qu'elle met en relation
- Généralement nommé avec un verbe
- Elle peut avoir une propriété ou pas.

Caractéristiques :

- Représentée par un ovale ou un hexagone
- Possède un nom (verbe à l'infinitif ou expression)
- Peut avoir des propriétés propres

Exemple :

Un client passe des commandes.



4.1.5. Etape 4 : Les Cardinalités

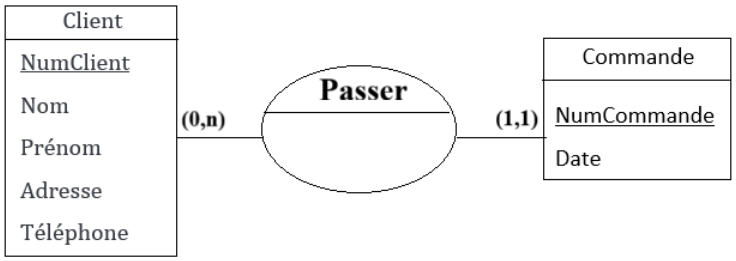
Les **cardinalités** définissent le nombre minimum et maximum de fois qu'une occurrence d'une entité peut participer à une association.

Notation : (min, max)

Les cardinalités courantes :

- **(0,1)** : zéro ou une fois (participation optionnelle unique)
- **(1,1)** : exactement une fois (participation obligatoire unique)
- **(0,n)** : zéro ou plusieurs fois (participation optionnelle multiple)
- **(1,n)** : au moins une fois (participation obligatoire multiple)

Exemple :



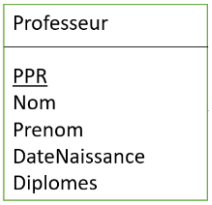
Interprétation : Un client peut passer 0 ou plusieurs commandes. Une commande est passée par exactement 1 client.

4.1.6. Etape 5 : Normalisation

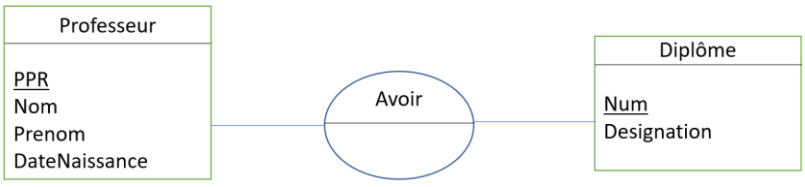
1ère forme normale (1FN):

- Toutes les entités doivent avoir un identifiant,
- Toute propriété doit être élémentaire (atomique). Aucune propriété n'est à valeurs multiples (ne contienne qu'une seule information à la fois).

Exemples :



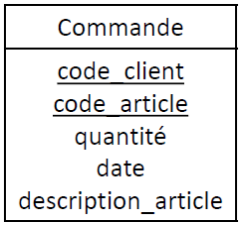
L'attribut diplomes n'est pas atomique, on va le séparer :



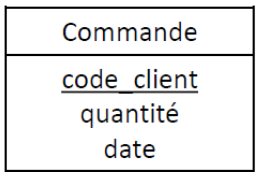
2ème forme normale (2FN):

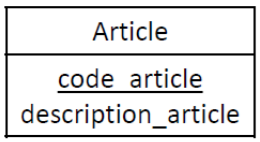
- Elle est en 1FN,
- Toutes les dépendances fonctionnelles entre la clé primaire et les autres attributs de la relation sont élémentaires. Autrement dit, les attributs doivent dépendre de la totalité de la clé.

Exemple :



Décomposer l'entité en deux entités :

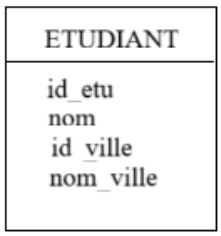




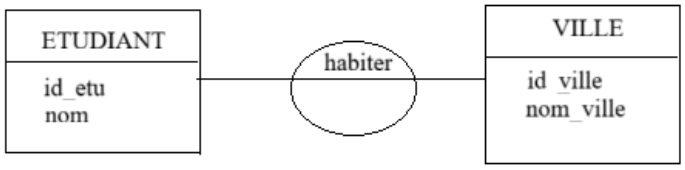
3ème forme normale (2FN):

- Elle est en 2FN,
- Aucune dépendance transitive entre attributs non clés.Un attribut non clé ne doit pas dépendre d'un autre attribut non clé.

Exemple :



Devient :



4.2. Du MCD au Modèle Logique de Données (MLD)

Le passage du MCD au MLD consiste à traduire le modèle conceptuel, qui décrit les données sans aucune contrainte technique, vers un modèle logique, qui prépare leur implémentation dans un SGBD relationnel (comme MySQL, Oracle, PostgreSQL…).

Le MLD reste indépendant du SGBD choisi, mais il respecte les règles du modèle relationnel.

Règles de Transformation :

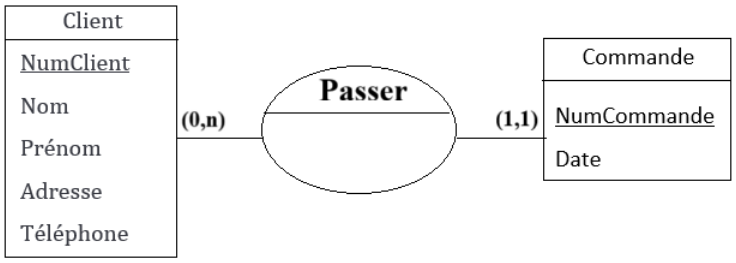
Règle 1 : Toute entité devient une table

- Les propriétés deviennent des colonnes
- L'identifiant devient la clé primaire

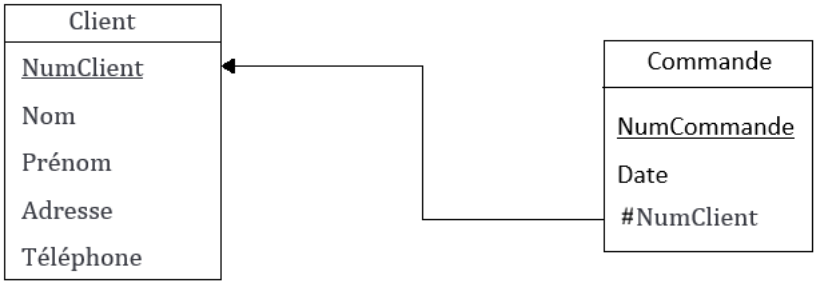
Règle 2 : Association avec cardinalités (x,1) - (x,n)

- La clé primaire du côté (x,1) migre vers le côté (x,n)
- Elle devient clé étrangère

Exemple :



Devient :



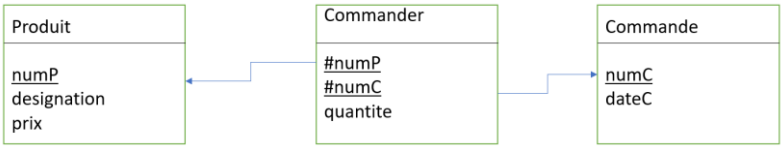
→ Table COMMANDE contient NumClient en clé étrangère

Règle 3 : Association avec cardinalités (x,n) - (x,n)

- L'association devient une table intermédiaire
- Elle contient les clés primaires des deux entités
- Ces clés forment une clé primaire composée

Exemple :

Devient :



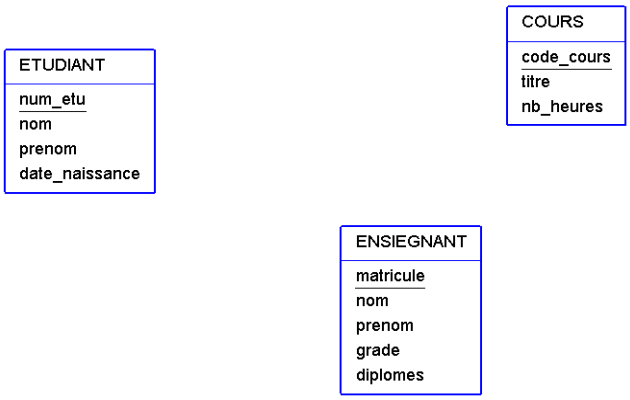
→ Table COMMANDER (NumP, numC, Quantite)

Exercice :

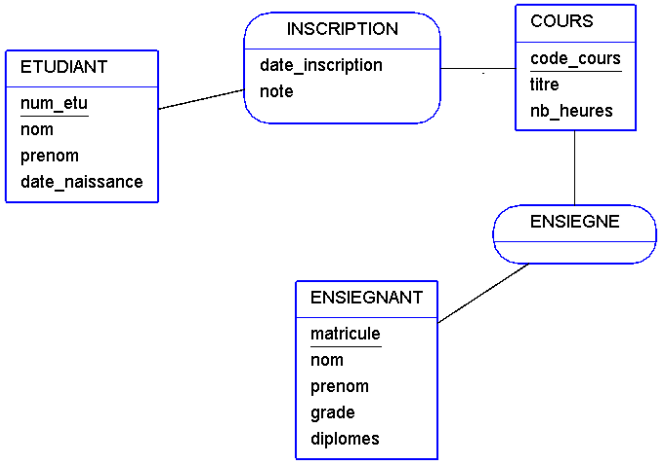
Une école supérieure souhaite gérer les étudiants, les cours et leurs inscriptions.
Les besoins exprimés sont les suivants :

- Chaque étudiant possède un numéro unique, un nom, un prénom, et une date de naissance.
- Chaque cours possède un code unique, un titre, et un nombre d'heures.
- Un étudiant peut suivre plusieurs cours, et un cours peut être suivi par plusieurs étudiants.
- Lorsqu'un étudiant s'inscrit à un cours, on enregistre la date d'inscription et la note obtenue.
- Chaque cours est assuré par un seul enseignant, mais un enseignant peut enseigner plusieurs cours.
- Un enseignant a un matricule unique, un nom, un prénom, un grade et ses diplômes.

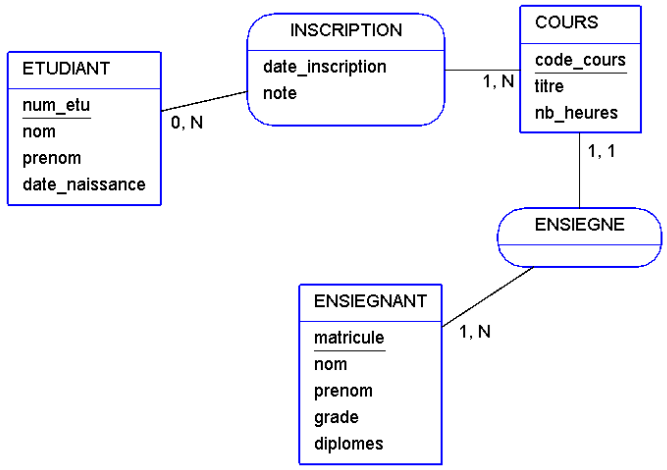
Etape 1 et 2 : Identifier les entités, attributs et identifiant



Etape 3 : Identifier associations

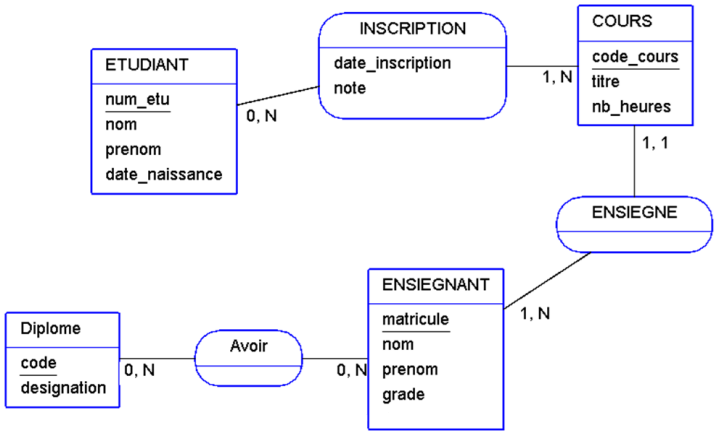


Etape 4 : Identifier les cardinalités



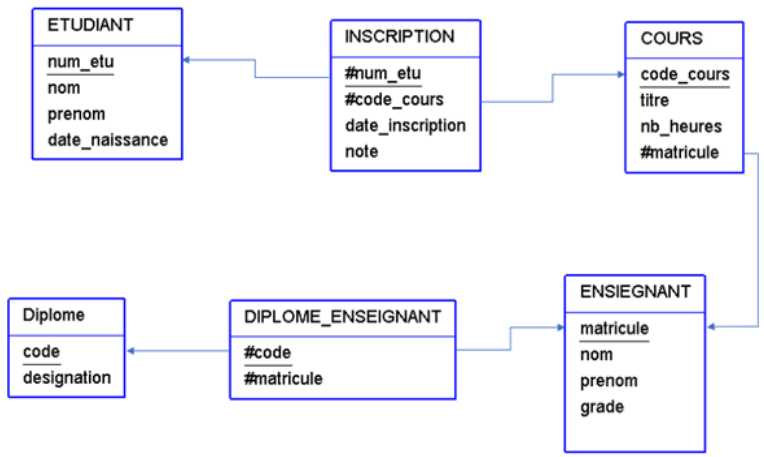
Etape 5 : Normalisation

L'entité ENSEIGNANT n'est pas en 1FN :



Conversion en MLD :

Nous allons changer le nom de l'association « Avoir » par « DIPLOME_ENSEIGNANT ».



Le schéma relationnel de la base de données :

```
ETUDIANT (num_etu, nom, prenom, date_naissance)
COURS (code_cours, titre, nb_heures, #matricule)
ENSIEGNANT (matricule, nom, prenom, grade)
DIPLOME (code, designation)
DIPLOME_ENSEIGNANT (#code, #matricule)
INSCRIPTION (#num_etu, #code_cours, date_inscription, note)
```

III. Le langage SQL (Structured Query Langage)

1. Un peu d'histoire

Les premières bases de données hiérarchiques apparaissent au début des années 1960 et Charles Bachmann conçoit en 1965 la première architecture d'un système de gestion de bases de données (SGBD). En 1968 Pick crée le premier système d'exploitation incluant un SGBD.

Dans les années 1970 Edgard Codd est l'initiateur de l'approche relationnelle. Sur une idée d'Edgard Codd, l'informaticien Donald Chamberlain développe une base de données relationnelle. Ceci a donné naissance dans les années 1970 au SQL, un langage spécifique pour interroger les bases de données, plusieurs fois amélioré et commercialisé dès 1979. À l'origine, on trouve le langage SEQUEL (Structured English Query Language).

Son efficacité modifie la plupart des activités comme les échanges boursiers, la gestion des salaires et impacte également notre vie courante comme l'utilisation d'une carte bancaire ou d'un smartphone.

2. Langage SQL

En 1986, intervient la première normalisation par l'ANSI avec l'appellation SQL-86 adoptée l'année suivante par l'ISO (Organisation internationale de standardisation).

Le SQL, pour Structured Query Language (langage de requêtes structurées) permet aux utilisateurs de communiquer avec une base de données. C'est un langage déclaratif qui décrit ce que l'on souhaite obtenir à l'aide d'une requête. La manière de parvenir au résultat n'est pas précisée, c'est le SGBD qui détermine la meilleure façon, en termes de complexité algorithmique, d'obtenir le résultat.

La suite du cours va être consacrée à la description des requêtes qui peuvent être effectuées sur une base de données.

Parmi les différentes fonctionnalités SQL, il est possible de retrouver les catégories de commandes suivantes :

- DDL (Data Définition Language) qui permet de définir et de modifier le schéma d'une base de données relationnelle.
- DML (Data Manipulation Language) qui permet l'interrogation d'une base de données relationnelle.
- DCL (Data Contrôle Language) pour contrôler la sécurité et l'intégrité de la base de données relationnelle.

Seules l'interrogation et la manipulation des données seront abordées.

3. Le langage de manipulation des données DML

Dans cette première partie on va s'intéresser à la manipulation des données, autrement dit nous allons voir comment interroger nos bases de données (Chercher des données, ajouter des données, modifier des données, supprimer des données).

On distingue typiquement quatre types de commandes SQL de manipulation de données :

- SELECT : sélection de données dans une table ;
- INSERT : insertion de données dans une table ;
- DELETE : suppression de données d'une table ;
- UPDATE : mise à jour de données d'une table.

Pour assimiler l'utilisation des différentes requêtes de ce cours, on va utiliser le schéma de la base de données de gestion d'une bibliothèque suivante :

Livre (<u>NumLivre</u>, TitreLivre, AuteurLivre, GenreLivre, EditeurLivre, NombrePages)
Eleve (<u>NumEleve</u>, NomEleve, DateNaissance, LieuNaissance)
Emprunter (<u>NumLivre*, NumEleve*, DateEmprunt,</u> DateRetour)

3.1. La requête SELECT

La requête SELECT permet de créer des sous-ensembles de données que vous pouvez utiliser pour répondre à des questions spécifiques. Vous pouvez également vous en servir pour fournir des données à d'autres objets de base de données.

La requête SELECT est la requête la plus importante du langage SQL et que nous allons détailler dans ce cours.

Syntaxe de la requête SELECT :

Mots	Arguments
SELECT	Attribut(s)
FROM	Table(s)
WHERE	Condition(s) sur une ligne
GROUP BY	Attribut(s) de partitionnement
HAVING	Condition(s) de sélection sur un groupe de lignes
ORDER BY	Attribut(s) de tri

Les mots clés ne sont pas sensibles à la casse contrairement aux données.

a) La projection

Une projection permet d'extraire des colonnes spécifiées d'une table.

Une projection s'exprime à l'aide de SQL par la commande :

```
SELECT [DISTINCT] [colonne1 , colonne2 .....]
FROM  nomtable ;
```

Par défaut, SQL n'élimine pas les doubles à moins que cela soit explicitement demandé par le mot clé DISTINCT.

Exemple : Pour obtenir la liste des noms des élèves avec leur date de naissance de la table **ELEVE** il sera nécessaire d'écrire :

```
SELECT NomEleve, DateNaissance
FROM Eleve ;
```

Si l'on souhaite éliminer les doubles, il faudrait noter :

```
SELECT DISTINCT NomEleve, DateNaissance
FROM Eleve;
```

b) La sélection

On parle également à ce niveau de restriction. Une sélection s'exprime à l'aide de SQL par la commande :

```
SELECT *
FROM  nomtable
WHERE  condition ;
```

Le symbole * signifie que l'on affiche toutes les colonnes d'une table. La combinaison avec une projection s'effectue en remplaçant * par la liste des colonnes à projeter.

Exemple 1 : On désire connaître les caractéristiques du ou des élèves dont le nom est BOUJIDI.

```
SELECT *
FROM Eleve
WHERE  NomEleve = 'BOUJIDI';
```

La même requête avec l'affichage du NumEleve et de la date de naissance s'écrit :

```
SELECT NumEleve, DateNaissance
FROM ELEVE
WHERE  NomEleve = 'BOUJIDI';
```

De manière plus générale, la condition suivant la commande WHERE peut inclure différents opérateurs de comparaison et différents opérateurs booléens :

** les opérateurs de comparaison :*

SYMBOLE	SIGNIFICATION
=	Égal
< > ou !=	Différent
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal

** les opérateurs logiques :*

SYMBOLE	SIGNIFICATION
Between...AND...	Entre ... et ...
Like	Comme
IS NULL	Dont la valeur est nulle
IN	Compris dans la liste

Avec l'opérateur Like, il est possible d'utiliser des caractères "joker" comme % et _.

Le "joker" % remplace de 0 à n caractères quelconques.

Exemple :

Si l'on écrit après la commande **WHERE attribut Like 'F%'**, la condition portera sur toutes les valeurs de l'attribut dont la première lettre commence par F.

Le "joker" **_** remplace un caractère quelconque et un seul.

Exemple :

Si l'on écrit après la commande **WHERE attribut Like'F_ _'** , la condition portera sur toutes les valeurs de l'attribut dont la première lettre commence par F et ayant ensuite seulement 2 autres caractères.

** les opérateurs booléens :*

SYMBOLE	SIGNIFICATION
AND	ET
OR	OU
NOT	NON

L'opérateur NOT inverse la valeur du résultat. Il est ainsi possible d'écrire : NOT BETWEEN, NOT IN, NOT LIKE, IS NOT NULL,

À ces opérateurs booléens, il peut être ajouté des parenthèses pour indiquer l'ordre d'évaluation.

c) Le tri :

Il est possible de trier les résultats suivant l'ordre ascendant (mot clé : ASC) ou descendant (mot clé : DESC) d'une ou plusieurs colonnes.

La commande SQL sera la suivante :

ORDER BY {expression | position} [ASC | DESC] [,{expression | position} [ASC | DESC] ....]

Exemple :

On désire relever les élèves dont le nom est BOUJIDI en affichant le numéro d'élève et le lieu de naissance. Les données seront présentées dans l'ordre croissant des lieux de naissance. L'instruction SQL sera la suivante :

```
SELECT NumEleve, LieuNaissance
FROM Eleve
WHERE  NomEleve ='BOUJIDI'
ORDER BY LieuNaissance ;
```

Si l'on veut obtenir les mêmes données mais en affichant les lieux de naissance dans un ordre décroissant, l'instruction SQL peut être la suivante :

```
SELECT NumEleve, LieuNaissance
FROM Eleve
WHERE  NomEleve='BOUJIDI'
ORDER BY LieuNaissance DESC ;
```

d) La jointure

Cette opération consiste à joindre 2 tables avec un critère de sélection. Une sélection s'exprime à l'aide de SQL par la commande :

```
SELECT *
FROM  nomtable1, nomtable2
WHERE  nomtable1.clé=nomtable2.clé ;
```

Exemple : On veut joindre la table EMPRUNTER et la table ELEVE.

L'instruction SQL sera la suivante :

```
SELECT *
FROM  Eleve, Emprunter
WHERE  Eleve.NumEleve=Emprunter.NumEleve ;
```

Il est possible de renommer les tables par des alias. Cela permet notamment d'écrire plus rapidement le code pour réaliser des requêtes et de réduire les erreurs de frappe. Ainsi pour l'exemple précédent, il est possible d'écrire la même requête de la manière suivante :

```
SELECT *
FROM  Eleve C , Emprunter  E
WHERE  C.NumEleve=E.NumEleve ;
```

Avec le langage SQL, il est possible de combiner les différentes opérations, il est ainsi possible de donner la liste des livres (titre des livres, genre et éditeur) empruntés par les élèves avec le nom 'RADI'. Dans la table on fera également apparaître le NumEleve pour distinguer éventuellement les élèves ayant le même nom.

Les différentes instructions sont les suivantes :

```
SELECT C.NumEleve, TitreLivre, GenreLivre, EditeurLivre
FROM Eleve C, Emprunter E, Livre L
WHERE NomEleve='RADI'
AND C.NumEleve = E.NumEleve
AND E.NumLivre = L.NumLivre ;
```

e) Les fonctions de calcul et les regroupements

Les fonctions de calcul :

Il est possible d'effectuer dans une requête SQL, des calculs horizontaux sur des lignes en utilisant les opérateurs +, -, *, / aussi bien dans la commande SELECT que dans la commande WHERE. Les arguments des opérateurs sont des noms de colonnes de type numérique ou des constantes.

Exemple : on souhaite afficher la table EMPRUNTER avec des dates d'emprunte décalées de 2 jours. L'instruction SQL sera la suivante :

```
SELECT NumEleve, CodeLivre, DateEmprunt + 2
FROM  Emprunter ;
```

Exemple : on souhaite afficher le numéro des élèves qui ont rendu le livre après 20 jours d'emprunt.

```
SELECT NumEleve
FROM Emprunter
WHERE DateRetour > DateEmprunt +20;
```

Les fonctions agrégatives :

Elles permettent d'effectuer des calculs verticaux pour l'ensemble ou un sous-ensemble des valeurs d'une colonne.

Les fonctions principales sont les suivantes :

FONCTIONS	SYMBOLE
SUM	permet d'effectuer la somme des valeurs d'une colonne numérique
AVG	permet d'effectuer la moyenne des valeurs d'une colonne numérique
MAX	permet de rechercher la valeur maximale d'une colonne numérique
MIN	permet de rechercher la valeur minimale d'une colonne numérique
COUNT	permet de compter le nombre de valeurs d'une colonne

- Pour compter le nombre de lignes sélectionnées, la fonction COUNT doit être utilisée avec l'argument * (COUNT(*)).
- Pour compter le nombre de valeurs distinctes prises par une colonne, il faut indiquer l'argument DISTINCT suivi de l'argument considéré.

- Les fonction agrégatives dans des instructions portant sur l'ensemble d'une colonne

Il n'y a pas ici de difficultés particulières

Exemple 1 :

Vous souhaitez déterminer le nombre de Eleves.

```
SELECT COUNT(NumEleve) AS NbEleve
FROM ELEVE ;
```

Exemple 2 :

Vous cherchez à déterminer le total des pages de tous les livres.

```
SELECT SUM(NombrePages) AS SommePages
FROM Livre
```

Exemple 3:

On désire déterminer la moyenne des pages pour les livres. La requête sera la suivante :

```
SELECT AVG(NbrePages) AS MoyPage
FROM Livre ;
```

- Les fonction agrégatives dans des Instructions portant sur les sous-ensembles d'une colonne

- **Le partitionnement**

Il doit permettre d'effectuer des calculs statistiques pour chaque sous-ensemble de lignes vérifiant un même critère. Le partitionnement s'exprime en SQL par la commande GROUP BY suivie du nom des colonnes de partitionnement.

Exemple 1 :

Vous souhaitez déterminer le nombre d'emprunt pour chaque Eleve ayant emprunté un livre.

```
SELECT NumEleve, COUNT(NumEleve)
FROM EMPRUNTER
GROUP BY NumEleve ;
```

Exemple 2 :

Vous cherchez à déterminer le total des pages par éditeur

```
SELECT EditeurLivre, SUM(NombrePages)
FROM LIVRE
GROUP BY EditeurLivre ;
```

Exemple 3 :

Vous cherchez à déterminer la moyenne du nombre moyen de pages par éditeur.

```
SELECT EditeurLivre, AVG(NombrePages)
FROM LIVRE
GROUP BY EditeurLivre;
```

- **Les conditions sur des classes de lignes.**

Lorsqu'une condition de recherche doit être exprimée non pas sur les lignes (WHERE) d'une table mais sur les **classes de lignes** introduites par un partitionnement, il est nécessaire d'utiliser la commande **HAVING**.

Exemple : Vous cherchez à sélectionner et visualiser pour chaque éditeur le nombre moyen de pages proposées. Seuls les éditeurs proposant une moyenne du nombre de pages supérieure à 83,1 devront être visualisés.

```
SELECT EditeurLivre, AVG(NombrePages) as MoyPages
FROM LIVRE
GROUP BY EditeurLivre
HAVING MoyPages >83.1 ;
```

f) Les requêtes imbriquées

Elles sont appelées également sous-requêtes. Il s'agit d'une requête incorporée dans la commande WHERE ou HAVING d'une autre requête (requête principale). Cette dernière utilise les résultats de la sous-requête.
Certaines sous-requêtes permettent de remplacer les jointures. Les sous-requêtes renvoient une ou plusieurs valeurs.

- **La requête ne renvoie qu'une seule valeur.**

L'imbrication de la requête avec la sous-requête se fera avec un opérateur de comparaison (=, >, <=, …)

Exemple : Quel est le CodeLivre ayant le nombre de pages le plus élevé ?

```
SELECT CodeLivre
FROM Livre
WHERE
NombrePages = (SELECT Max(NombrePages)  FROM Livre );
```

Remarque : La sous-requête est notée entre parenthèses.

- **La sous-requête renvoie plusieurs valeurs**

L'imbrication se fera avec bien souvent l'opérateur logique IN

Exemple :
On souhaite connaître la liste des numéros des élèves qui ont emprunté un livre contenant plus que 120 pages
La requête SQL peut alors être la suivante :

```
SELECT NumEleve
FROM Emprunter, Livre
WHERE
Emprunter.NumLivre=Livre.NumLivre   AND   NombrePages>=120
```

Elle peut également être la suivante :

```
SELECT NumEleve
FROM EDITEUR
WHERE
NumLivre IN  (SELECT NumLivre FROM  Livre WHERE NombrePages>=120)
```

Dans la première formulation, la mention DISTINCT permet d'éliminer les doublons.

ORDER BY ne peut être noté dans une sous-requête.

3.2. La commande INSERT

Elle permet d'insérer un ou plusieurs tuples dans une table. Il est possible de dire également que cette commande va permettre d'assurer le "peuplement" des tables de la base à partir de données nouvelles.

Le formalisme est le suivant :

```
INSERT INTO <TABLE>
VALUES ( ' valeur1 ',' valeur2 ') ;
```

Exemple :

```
INSERT INTO EDITEUR
VALUES ('LACO1','BertrandLacopee', '8, rue de Nevers','75009','PARIS')
```

Si la mise à jour n'existe que pour quelques attributs, le formalisme est le suivant :

```
INSERT INTO <TABLE> (ATTRIBUT1, ATTRIBUT2)
VALUES ('valeur1','valeur2') ;
```

Il est alors nécessaire de respecter l'ordre des attributs.

Exemple :

On souhaite ajouter un livre. Deux éléments seulement sont connus le codelivre (F132) et le codeediteur (FOU1)

```
INSERT INTO LIVRE (CodeLivre, CodeEditeur)
VALUES ('F132','FOU1')
```

Dans tous les cas, les valeurs de la liste sont séparées par des virgules. Des guillemets simples doivent encadrer les données dès qu'elles comportent des caractères ou des dates. C'est inutile pour des données numériques ou pour des valeurs NULL (absence de valeurs).
Ainsi pour l'exemple précédent, il était également possible d'écrire :

```
INSERT INTO LIVRE
VALUES ('F132','','','FOU1')
```

Ou bien encore, il était possible d'écrire :

```
INSERT INTO LIVRE
VALUES ('F132',NULL,NULL,FOU1)
```

Remarque : L'absence de valeur pour un attribut (ou colonne) doit avoir été autorisée. C'est par exemple impossible pour la clé primaire.

3.3. La commande UPDATE

Elle permet de mettre à jour des données dans une table.

Le formalisme est le suivant :

```
UPDATE <table>
SET <attribut>=<' valeur '>
[WHERE <condition>] ;
```

Si le mot réservé WHERE est facultatif et permet de faire une mise à jour sous certaines conditions, il est très rare de ne pas le noter. L'inverse peut-être dangereux car l'ensemble de la colonne serait modifié.

Exemple : Le code éditeur a maintenant pour nom Foucher et Compagnie.

```
UPDATE EDITEUR
SET NomEditeur = 'Foucher et Compagnie'
WHERE CodeEditeur = 'FOU1'
```

S'il y a 2 mises à jour dans la même requête, il faut les séparer par une virgule et ne pas mettre 2 fois SET
La syntaxe est la suivante :

```
UPDATE <table>
SET     <attribut1>=<' valeur2 '>,
        <attribut2>=<' valeur4 '>
[WHERE <condition>] ;
```

Exemple : Le code éditeur a maintenant pour nom Fourcher et Compagnie et pour adresse 31 rue de Fleurus.

```
UPDATE EDITEUR
SET     NomEditeur = 'Fourcher et Compagnie'
        AdresseEditeur ='31, rue de Fleurus'
WHERE CodeEditeur = 'FOU1'
```

3.4. La commande DELETE

Elle permet d'effacer un ou plusieurs tuples.

```
DELETE FROM <table>
[WHERE <condition>] ;
```

Ici aussi le mot réservé WHERE est facultatif et permet de faire une suppression sous certaines conditions. Mais de la même manière, il est très rare de ne pas le noter. L'inverse peut-être dangereux car l'ensemble des enregistrements serait supprimé.

Exemple

On souhaite supprimer l'éditeur dont le code est F132.

La requête SQL est la suivante :

```
DELETE FROM EDITEUR
WHERE CodeEditeur ='F132'
```

IV. Reporting et Applications No-Code

1. Introduction : Pourquoi le reporting ?

Dans toute organisation (entreprise, école, hôpital, administration), les données sont partout : ventes, stocks, résultats, fréquentation, satisfaction, etc.

Mais la donnée brute ne sert à rien si elle n'est pas transformée en information utile. C'est le rôle du reporting : collecter, organiser et présenter les données pour aider à la décision.

Exemple : un tableau de bord de suivi des étudiants peut montrer :

- le nombre d'inscrits par filière,
- le taux de réussite,
- la moyenne générale par module,
- les enseignants les plus sollicités.

2. Qu'est-ce que le reporting ?

2.1. Définition

Le reporting est l'ensemble des outils et méthodes qui permettent de présenter des informations synthétiques et visuelles issues de la base de données.

Objectif principal :

Transformer les données en indicateurs de pilotage (KPI) compréhensibles et exploitables.

Type	Description	Exemple
Opérationnel	Suivi quotidien de l'activité	Nombre d'inscriptions par jour
Tactique	Analyse à moyen terme	Moyenne des notes par enseignant
Stratégique	Indicateurs de performance globaux	Taux de réussite annuel par filière

2.2. Processus de reporting

1. Collecte des données :
   Depuis la base de données, un fichier Excel, ou un ERP.

2. Nettoyage et préparation
   Vérification des doublons, formats, et cohérence des valeurs.

3. Analyse et calculs
   Utilisation de requêtes SQL ou d'outils analytiques pour obtenir les KPI.

4. Visualisation et diffusion
   Création de graphiques, tableaux croisés et tableaux de bord interactifs.

2.3. Outils de reporting modernes

Outil	Type	Points forts
Excel / Power Query / Power Pivot	Tableur analytique	Simple, universel, parfait pour débuter
Power BI (Microsoft)	BI (Business Intelligence)	Rapports dynamiques, visuels interactifs
Google Data Studio / Looker Studio	BI Cloud	Rapports en ligne et collaboratifs
Tableau Software	BI professionnel	Grande puissance visuelle et analytique
Metabase, Superset, Redash	Open Source BI	Pour entreprises ou projets étudiants

3. Introduction aux applications No-Code

3.1. Définition

Les applications no-code permettent de créer des applications web ou mobiles sans écrire de code, grâce à une interface visuelle (glisser-déposer).

Objectif

Donner aux utilisateurs métier la capacité de créer leurs propres outils :

- formulaires de saisie,
- tableaux de suivi,
- mini-CRM,
- gestion de stock,
- interface d'exploitation d'une base existante.

3.2. Outils No-Code

Outil	Type	Exemple d'usage
Airtable	Base de données + interface	Suivi d'étudiants ou projets
Glide Apps	Création d'applis mobiles à partir d'un Google Sheet	Application de suivi de notes
AppSheet (Google)	Applications métier connectées à Sheets ou SQL	Suivi de commandes, formulaires
Notion + Automate.io / Make	Documentation + automatisation	Portail de suivi collaboratif
Bubble	Création d'applis web avancées	Mini portail web avec authentification