# Analyse et Conception de Base de Données

## INTRODUCTION :

Une base de données est pour certains une collection de fichiers reliés par des pointeurs multiples, aussi cohérents entre eux que possible, organisés de manière à répondre efficacement à une grande variété de question. Pour d'autres, une base de données peut apparaître comme une collection d'information modélisant une entreprise du monde réel.

### II) Concepts Analyse et Conception

Le niveau central est le niveau conceptuel. Il correspond à la structure sémantique inhérente sans souci d'implémentation en machine, représentant la vue intégrée de tous les utilisateurs.

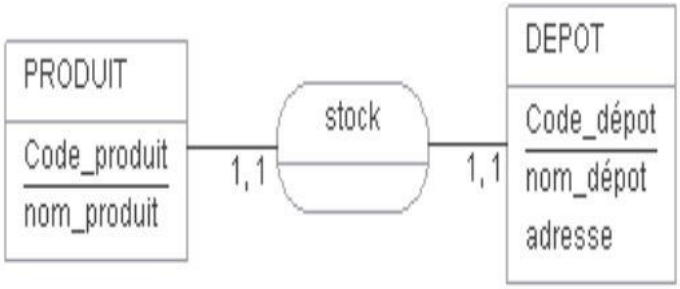
#### A) MCD

Une **entité** est définie comme un **objet** pouvant être identifié distinctement. Cet objet peut représenter des individus (un client), des objets (un libre) ou, une abstraction (compte bancaire) ayant une existence propre ou des événements (une commande). Le modèle conceptuel comprend des entités types (par exemple le client), représentant des ensembles d'entités de la réalité (les clients de l'entreprise). Les entités sont décrites par des **attributs**, caractéristiques ou propriétaires. Par exemple le client a un code, une raison sociale, une adresse, un montant de découverte, etc… Parmi tous les attributs de l'entité, on définit un **identifiant**, qui est un attribut ou un ensemble d'attributs permettant de déterminer une et une seule entité à l'intérieur de l'ensemble. Ainsi le code_client identifie un seul client de l'entreprise. Les **relations** représentent les liens existants entre entités. Ainsi les clients sont reliés aux commandes. Au niveau du modèle, on dit qu'il existe une relation entre le client et la commande. Dans la réalité, il n'y a un ensemble de relation deux à deux entre commande et client. Contrairement aux entités, les relations n'ont pas d'existence propre. Elles sont caractérisées, comme les entités, par un nom et un éventuellement des attributs. Le nombre d'entités impliquées dans une relation est appelé **dimension** ou **degré** de la relation. Ainsi, une relation entre la commande et le client est binaire ou de dimension deux. Plus généralement une relation peut être de :

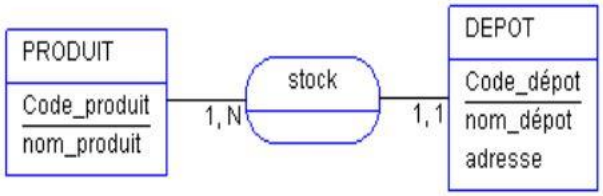
- **dimension 1** : dans ce cas elle ne concerne qu'une entité type dont elle relie deux éléments. Elle est dite réflexive. Par exemple, La relation Mariage relie deux élément de l'entité type PERSONNE ;
- **dimension 2** :C'est le cas le plus fréquent ;
- **dimension 3** : Par exemple, une location de véhicule représente une relation entre un véhicule, une personne et une date. Cette relation est ternaire dans la mesure où elle peut être décomposée en deux ou trois relations binaires.

De façon générale, une relation peut être caractérisée par n dimensions. La description complète d'une relation nécessite la définition précise de la participation des entités d'une entité dans une relation. On appelle **cardinalité** le nombre de participation d'une entité dans une relation. Ainsi une relation peut être de cardinalité **1-1**, **1-N** ou **M-N**. Par exemple considérons la relation binaire stock entre deux ensembles d'entités PRODUIT et DEPOT. La relation est de cardinalité :

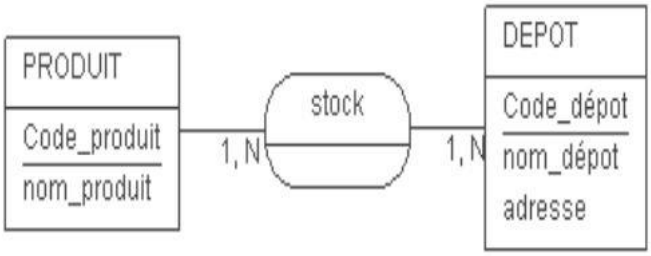
- **1-1** si et seulement si un produit ne peut être stocké que dans un seul dépôt et un dépôt ne contient qu'un seul type de produit ;



• **1-N** dans le cas où un produit peut être stocké dans plusieurs dépôts mais où chaque dépôt ne contient qu'un seul type de produit, ou inversement (N-1)




• **M-N** si un type de produit est stocker dans plusieurs dépôts et un dépôt donné peut contenir plusieurs types de produits.




Pour les relations ternaires les cardinalités possibles sont **(1-1-1), (1-1-N), (1-M-N)** et **(MN-P)**. Ces cardinalités traduisent les règles de gestion de l'entreprise ou les contraintes propres aux entités et relation. Elles sont appelés **cardinalités maximales** dans la mesure où elle représente le nombre maximum de participation d'une entité à une relation.
Les concepts ci-avant constituent les éléments du modèle (MCD) de base. Celui-ci peut être étendu en y intégrant les notions de cardinalité minimale et de généralisation d'entité. Dans ce cas on le modèle devient modèle MCD étendu (ou enrichi).
Dans le modèle MCD les entités sont représentées par des rectangles et les associations entre entité par des ellipses. Les attributs identifiants des entités sont soulignés. On distingue deux types d'entités : les entités faibles et les entités régulières. Une entité est dite faible si son existence dépendant de l'existence d'une autre entité. Par exemple, une entité LIGNE_COMMANDE n'existe que si l'entité COMMANDE correspondante est présente. Les autres entités sont dites régulières.

### ❖ Formalisme de Représentation

A toute situation à modéliser, peuvent correspondre plusieurs schéma différents avec leurs avantages et leurs inconvénients. Il est souvent peu aisé de dégager une modélisation largement supérieure aux autres. D'où les critères suivants pour mesurer la qualité d'une modélisation MCD:
• L'expressivité : elle traduit la richesse sémantique du schéma. Elle peut être traduite par le nombre de concepts et/ou de contraintes exprimés dans le schéma ;
• La minimalité : elle tend à privilégier les schémas introduisant un nombre de redondances minimales ;
• La lisibilité : Elle consiste à évaluer la représentation graphique proprement dite, par exemple, en préférant un schéma où un minimum d'arcs se croisent ;
• La simplicité : elle privilégie les schémas contenant un nombre de concepts minimum
La construction d'un schéma conceptuel peut se réaliser de la manière suivante :

a) Déterminer la liste des entités.
b) Pour chaque entité :
a. établir la liste de ses attributs ;
b. parmi ceux-ci, déterminer un identifiant.
c) Déterminer les relations entre les entités.

d) Pour chaque relation :
a. dresser la liste des attributs propres à la relation ;
b. vérifier la dimension (binaire, ternaire, etc.) ;
c. définir les cardinalités.
e) Vérifier le schéma obtenu, notamment :
a. supprimer les transitivités ;
b. s'assurer que le schéma est connexe ;
c. s'assurer qu'il répond aux demandes.
f) Valider avec les utilisateurs.

❖ **Les erreurs à ne pas commettre**

Nombre d'erreurs sont fréquemment commises lors du développement d'un schéma conceptuel.
a) Surestimer la dimension d'une relation. La relation est ternaire si la quantité commandée est la quantité totale. Elle est quaternaire si l'on veut enregistrer toutes les commandes passées par un client.
b) Attribuer à une relation les attributs des entités participantes ou inversement

c) Exprimer des relations redondantes, c'est-à-dire déductible par transitivité
d) Se tromper du niveau de discours
e) Confondre les concepts de données et celui de traitement

## B) MLD

Le **MLD** ou Modèle Logique des Données est simplement la représentation textuelle du **MPD**. Il s'agit juste de la représentation en ligne du schéma représentant la structure de la base de données. Il n'y a pas de travail poussé à réaliser à cette étape, il s'agit juste d'appliquer quelques règles toutes simples. Cette étape est parfois omise.

### ● MLDR

On représente ainsi les données issues de la modélisation Merise sous la forme suivante :

- ✓ Chaque ligne représente une table ;
- ✓ C'est toujours le nom de la table qui est écrit en premier ;
- ✓ Les champs sont listés entre parenthèses et séparés par des virgules ;
- ✓ Les clés primaires sont soulignées et placées au début de la liste des champs ;
- ✓ Les clés étrangères sont préfixées par un dièse.

À noter que le MLD prend parfois un R et devient MLDR : le R signifiant simplement Relationnel.

### ● Modèle Relationnel

Dans le modèle relationnel, les entités du schéma conceptuel sont transformées en tableaux à deux dimensions.
Il existe trois types de clés:

- **Clé primaire** : Ensemble minimum d'attributs qui permet de distinguer chaque n-uplet de la table par rapport à tous les autres. Chaque table doit avoir une clé primaire.
- **Clé candidate** : Ensemble minimum d'attributs susceptibles de jouer le rôle de la clé primaire.
- **Clé étrangère** : fait référence à la clé primaire d'une autre table.

#### 1) Démarche

Les règles principales de transformation d'un schéma conceptuel **MCD au MLDR** relationnel sont :
**Règle 1** : Toute entité est traduite en une table relationnelle dont les caractéristiques sont les suivantes :
• le nom de la table est le nom de l'entité ;
• la clé de la table est l'identifiant de l'entité ;
• les autres attributs de la table forment les autres colonnes de la table.
**Règle 2** : Toute relation binaire (N-N) est traduite en une table relationnelle dont les caractéristiques sont les suivantes :

- le nom de la table est le nom de la relation ;
- la clé de la table est formée par la concaténation des identifiants des entités participant à la relation ;
- les attributs spécifiques de la relation forment les autres colonnes de la table.

Une contrainte d'intégrité référentielle est générée entre chaque colonne clé de la nouvelle table et la table d'origine de cette clé.

**Règle 3** : Toute relation binaire (1-N) est traduite :

- soit par un report de clé : l'identifiant de l'entité participant à la relation côté N est ajoutée comme colonne supplémentaire à la table représentant l'autre entité. Cette colonne est parfois appelée *clé étrangère*. Le cas échéant, les attributs spécifiques à la relation sont eux aussi ajoutés à la même table ;
- soit par une table spécifique dont les caractéristiques sont les suivantes :
  - le nom de la table est le nom de la relation ;
  - la clé de la table est l'identifiant de l'entité participent à la relation côté 1 ;
  - les attributs spécifiques de la relation forment les autres colonnes de la table

**Règle 4** : Toute relation binaire (1-1) est traduite, au choix, par l'une des trois solutions suivantes :

- fusion des tables des entités qu'elle relie (choix1) ;
- report de clé d'une table dans l'autre (choix2) ;
- création d'une table spécifique reliant les clés des deux entités (choix3).

Les attributs spécifiques de cette relation sont ajoutés à la table résultant de la fusion (choix1), reportés avec la clé (choix2), ou insérés dans la table spécifique (choix3).

**Règle 5 :** Toute relation ternaire est traduite par une table spécifique, sauf cas particulier, la clé de cette table est constituée par la concaténation des identifiants des entités participant à la relation. Les attributs spécifiques de la relation constituent les autres colonnes de la table.

**Règle 6** : Toute généralisation d'entité E de n entités E1, E2, …, En peut être traduite au choix par l'une des trois solutions :

- la création d'une seule table représentant l'entité Générique E et intégrant tous les attributs des entités spécifiques. Les relations éventuelles impliquant ces entités sont alors considérées comme impliquant l'entité Générique E avec une cardinalité minimale nulle ;
- La création de n tables représentant les entités E1, E2, …, En qui héritent de l'ensemble des attributs et relation de l'entité générique E ;
- La création conjointe des tables E, E1, E2, …, En.

**Règle 7** : Toute relation récursive est considérée comme une relation binaire. Sa traduction dépend donc du type de cette relation binaire (1-1, 1-N ou M-N) et obéit à l'une des règles 2, 3 ou 4

### 2) Erreurs à ne pas commettre

- Le sens de report des clés pour les relations 1-N
- Oubli du report des attributs spécifiques des relations 1-N
- Difficulté de l'identification des ternaires et plus
- Répercussion des erreurs de modélisation conceptuelle

## IV ) SQL

### 1) Introduction

SQL (Structured Query Language) est un langage de requêtes standard pour l'interrogation de bases de données relationnelles. Première version SQL-1 en 1989, puis SQL-2 en 1992, SQL-3 ?

Il est développé à l'origine pour le prototype de SGBD recherche d'IBM SYSTEM/R, qui a débouché sur les produits commerciaux SQL/DS et DB-2. C'est un mélange d'algèbre relationnelle et de calcul relationnel à variables n-uplets

### 2) Les concepts

#### a. Le langage de définition des données

Dans le langage de description de données, nous décrivons les instructions SQL permettant de créer, de modifier, de supprimer ou de renommer les éléments constitutifs d'un schéma de base de données relationnelle : les tables, les vue et les index.

### i. La création de la table

La création d'une table peut être réaliser au moyen de la commande suivante :

**CREATE TABLE nom_table (nom_col1 type_col1 [DEFAULT valeur1] [contrainte_col1], nom_col2 type_col2 [DEFAULT valeur2] [contrainte_col2],… contrainte_table1, contrainte_table2,…) ;**

En majuscule figurent les mots réservés du langage , en minuscule les noms des objets de la base de données (colonne, tables, contraintes,…). Entre crochets figurent les éléments facultatifs.

Une table doit avoir au moins une colonne. Chaque colonne porte un nom qui est unique dans la table. Type_col1, type_col2, etc. représente les types de données (caractère, numérique, date, etc.). Les types des colonnes sont à choisir parmi les types fournis par le SGBD. La clause DEFAULT permet de préciser, le cas échéant, une valeur par défaut qui sera insérée en cas d'absence de valeur pour cette colonne.

Les contraintes de colonne, c'est-à-dire ne portant que sur un attribut, sont spécifiées immédiatement après chaque colonne alors que les contraintes de table, spécifiées à la fin de l'instruction, porte généralement sur plusieurs colonnes de la table. La définition d'une contrainte peut être suivie d'un nom (CONSTRAIN nom_contrainte). Contrainte de colonne et contraintes de table matérialise les différentes contraintes d'intégrité dont le SGBD prend en charge la vérification systématique:

- La contrainte d'unicité (clause UNIQUE) permet d'assurer qu'il n'existe pas de valeur dupliquée dans la colonne. En d'autres termes, la colonne est une clé de la table. Si la clause UNIQUE porte sur plusieurs tables, alors elle est une contrainte de table ;
- La contrainte d'obligation (clause NOT NULL) permet de vérifier qu'une colonne ne contient pas de valeurs manquantes. En d'autres termes, elle interdit l'insertion de données pour lesquelles cette colonne n'aura pas de valeur. NOT NULL est toujours une contrainte de colonne, et ne peut pas être une contrainte de table.
- La contrainte clé primaire (clause PRIMARY KEY) permet de choisir une colonne (ou un groupe de colonnes) unique privilégiée dans une table. Autrement dit, elle a le même rôle que la clause UNIQUE mais elle ne peut être spécifiée qu'une seule fois pour une table, alors que l'on peut avoir une (ou groupe de colonnes) portant la clause UNIQUE. La clause PRIMARY KEY interdit l'absence de valeur (elle induit automatiquement une contrainte NOT NULL). Elle est la colonne privilégiée pour l'expression des contraintes référentielles ;
- La contrainte d'intégrité référentielle admet deux syntaxes selon qu'elle porte sur une colonne (contrainte colonne) ou sur plusieurs colonnes (contrainte table). Dans le premier cas, on utilise la clause REFERENCES. Dans le second cas on utilise la clause FOREIGN KEY. Elle matérialise une dépendance d'inclusion entre deux colonnes appartenant à deux tables différentes ou deux groupes de colonnes. Ainsi, on pourra stipuler une contrainte référentielle entre la colonne code_client de la table COMMANDE et la colonne code_client de la table CLIENT pour s'assurer que toute commande correspond bien à un client déjà enregistré. Par défaut, la contrainte référentielle référence la clé primaire de la table visée.
- La contrainte dite sémantique (clause CHECK) permet de spécifier des conditions logiques portant sur une ou plusieurs colonnes d'une même table. Elle est utilisée pour de simples contrôles de domaine, par exemple un salaire est compris entre deuxvaleurs extrêmes, ou pour des vérifications plus complexes, par exemple, l'inclusion d'une colonne dans une liste de valeurs, ou la comparaison des valeurs de différentes colonnes de la table, tels le prix de vente et le prix d'achat d'un produit.

### ii. Modification de données

La commande ALTER TABLE permet la modification de la structure d'une table : ajout d'une colonne, ajout d'une contrainte, modification de la définition d'une colonne, suppression d'une contrainte, etc. La syntaxe est la suivante :

**ALTER TABLE nom_table modification;**

Les possibilités de modification de la structure d'une table dépendent largement des SGBD.

On peut ajouter (ADD COLUMN) ou supprimer une colonne (DROP COLUMN), modifier les éléments de définition d'une colonne (ALTER COLUMN), ajouter (ADD CONSTRAINT nom_contrainte) ou supprimer

une contrainte (DROP CONSTRAINT nom_contrainte).
Par exemple, la commande :
**ALTER TABLE produit ALTER COLUMN libellé CHAR(30) ;**
permet d'étendre la taille d'une colonne à trente caractères.
De la même façon, on peut supprimer une table à l'aide de la commande :
**DROP TABLE nom_table ;**
On peut renommer une table avec la commande :
**RENAME ancien_nom TO nouveau_nom ;**

**TRUNCATE TABLE** est utilisée pour supprimer tous les enregistrements d'une table. Elle exécute la même fonction qu'une instruction DELETE sans clause WHERE. Avertissement: si vous tronquez une table, l'instruction TRUNCATE TABLE ne peut pas être annulée dans certaines bases de données.

### b. Le langage d'interrogation des données

Aussi appelé langage de manipulation des données, il comprend quatre instructions principales :

- **Jointures croisées et non restreintes**

Une jointure croisée ou non restreinte produit un ensemble de résultats qui comprend toutes les combinaisons de toutes les lignes entre les tables de la jointure. C'est en fait le produit cartésien des tables jointes. Par exemple si une table présente 8 lignes et une autre 9 lignes, le résultat 72 lignes.

- **La jointure externe**

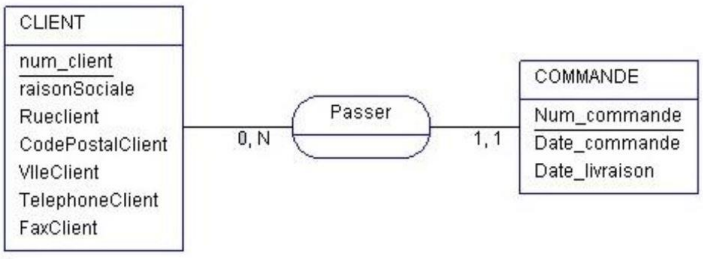
Dans le cas d'une jointure interne ou croisée, lorsqu'une ligne d'une table ne satisfait pas à la condition de jointure, cette ligne n'apparaît dans le résultat final de la requête.
Il peut cependant être souhaitable de conserver les lignes d'une table qui ne répondent pas à la condition de jointure. On parle alors de semi-jointure ou de jointure externe.
On distingue 3 types de jointures externes :

- jointure externe gauche ou semi-jointure gauche : inclut toutes les lignes issues de la première table nommée.
- jointure externe droite ou semi-jointure droite inclut toutes les lignes issues de la deuxième table nommée.
- jointure externe intétragrale : inclut les lignes sans correspondance des tables droites et gauches

#### i. jointure externe gauche et droite

Les jointures externes droite et gauche peuvent être créées en employant les opérations de jointure externe droite << * => et jointure externe droite << = *>> dans la clause WHERE d'une instruction SELECT.

Exemple




#### ii. jointure externes intégrales ou jointure généralisée (FULL OUTER JOIN)

Les jointures externes intégrales sont utilisées lorsque deux tables sont jointes et que chacune contient plusieurs lignes qui ne correspondent à aucune ligne dans l'autre table. Elle consiste à faire apparaître les lignes des deux tables qui ne satisfont pas à la condition de jointure. La réalisation de la jointure externe

intégrale peut se faire en utilisant l'union de la semi jointure droite et de semi-jointure gauche.
SELECT * FROM table1, table2 WHERE colonne1*=colonne2 UNION SELECT * FROM table1, table2 WHERE colonne1=*colonne2;

**Remarque**:
Suivant la base de données utilisée, d'autres syntaxes sont possibles pour les jointures externes

**iii. Inequijointure**

La mise en relation des colonnes communes à deux tables ne s'établit pas forcément par l'intermédiaire d'une opération d'égalité. On parle alors d'inéquijointure. Les opérations possibles sont : **>, <, <=,>= et** <>

**iv. L'auto jointure ou jointure d'une table sur elle-même**

Une auto jointure relie les lignes d'une table avec d'autre ligne de la même table.
Par exemple, la recherche de tous les clients ayant passé une commande à la même date que la commande 132. En SQL cette requête s'écrira :

**SELECT C1.numcli FROM commande C1, commande C2 WHERE C1.date_commande=C2.dte_commande AND C2.num_com=132;**

Pour joindre une table à elle-même, vous devez lui affecter deux alias (abréviation de table) afin que la table puisse être référencée comme deux tables distinctes. Les alias sont affectés dans la clause FROM en spécifiant l'alias par le nom de la table.
L'utilisation d'un alias permet de renommer une des tables et évite les problèmes d'ambiguïté pour les noms de colonnes qui doivent être préfixées par les synonymes des différentes tables.

- **La projection**

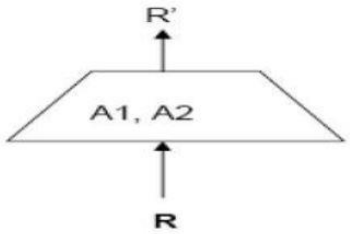
La projection d'une relation R consiste à créer une nouvelle relation, à partir de R mais en ne conservant que les attributs cités en opérande. Si nous nous plaçons du côté utilisateurs, cela veut dire que parmi les attributs constituants les tuples, les valeurs de certains ne nous intéressent pas temporairement pour l'objectif à atteindre.

**Définition**

**La projection (proj ou Π) est l'opération qui consiste à :**

- Supprimer, d'une relation, les attributs non mentionnés en opérande,
- Et à éliminer les tuples, en doublon, qui risquent d'apparaître dans la nouvelle table.

Soit un schéma de relation R (A1, A2… An), avec ∀ i ∈ E (entiers), Ai étant un attribut dont les valeurs appartiennent à un domaine Di.
La projection R' de R sur A1, A2 s'écrira :**R' = proj (R, A1, A2) = Π A1, A2 (R)**
La modélisation graphique suivante est aussi possible :




**Pourquoi faut-il éliminer les tuples en doublon qui risquent d'apparaître ?**
Si la clé primaire (constituée d'une composition minimale d'attribut(s)) n'apparaît pas dans les opérandes de la projection, **la relation résultante n'aura pas de clé primaire définie, d'où risque de tuples en double**. De ce fait, une clé primaire est définie arbitrairement dans le cas de la projection. Ce sera la composition de tous les attributs restants dans cette relation : ce qui permet d'éliminer tout doublon dans les tuples.

- l'instruction SELECT pour l'interrogation d'une ou plusieurs tables ;

Syntaxe :
**SELECT liste_colonnes FROM liste_tables**
**[WHERE condition]**

**[ORDER BY liste_colonnes]**
**[GROUP BY liste_colonnes]**
**[HAVING condition_groupe]**
La clause **SELECT** sélectionne les colonnes mentionnées dans la liste, provenant d'une des tables précisées dans la clause **FROM**. La requête affiche toutes les lignes des tables vérifiant la condition exprimée dans la clause **WHERE.** En l'absence de clause **WHERE**, toutes les lignes sont affichées. La clause **ORDER BY** est facultative et permet de trier le résultat d'une requête.
La clause S**ELECT** peut comprendre ou combiner les colonnes des tables, des expressions calculées sur ces colonnes ou encore des chaines de caractères spécifiées entre cote (‘). La clause FROM décrit la table ou les tables interrogées. Par exemple :
**SELECT** nom **FROM** client ;
Liste l'ensemble des noms de clients contenus dans la table.
**SELECT ‘Marge :’, prix_vente-prix_achat FROM produit ;**
Illustre la possibilité d'insérer des chaines de caractères et des expressions calculées dans la clause **SELECT.**
Pour éviter la répétition dans l'affichage d'une colonne on utilise la clause DISTINCT qui supprime les doublons.
**SELECT DISTINCT nom FROM client;**
Si l'on veut afficher toutes les colonnes d'une table on utilise l'astérisque (*****).
**SELCET * FROM client ;**
La clause **WHERE** permet un affichage sélectif des lignes d'une table. Elle est suivie d'une expression logique.
L'expression logique peut être simple ou composée de plusieurs éléments reliés par des opérations logiques **AND, OR et NOT.**
La requête : **SELECT num_client FROM client WHERE ville=’Londres’ ;**
Recherche et affiche les numéros de client clients localisés à Londres.
La requête : **SELECT num_client FROM client WHERE ville=’Londres’ AND chiffre_affaire> 10000 ;**
se limite aux client de Londres dont le chiffre d'affaire est supérieur à un seuil.
Si l'on souhaite afficher les clients localisés à Londres ou à Paris, on écrira :
**SELECT num_client FROM client WHERE ville=’Londres’ OR ville=’Paris’ ;**
**La requête SELECT num_client FROM client WHERE ville <> Londres AND chiffre_affaire<=5000 ;**
Permet de rechercher les numéros des clients extérieurs à Londres dont leur chiffre d'affaire est inférieur à 5000.
Pour exprimer une inclusion dans un intervalle de valeur, on peut au choix faire une double comparaison ou utiliser l'opération **BETWEEN.**
Exemple:
**SELECT num_client FROM client WHERE chiffre_affaire<=5000 AND chiffre_affaire<=10000;**
On peut aussi tester l'existence d'une valeur avec l'opérateur IS NULL ou son absence avec l'opérateur IS NOT NULL.
**SELECT num_client FROM client WHERE ville IS NULL;**
Recherche les numéros des clients dont la ville est inconnue. Pour analyser les contenus chaînes de caractères, on dispose de l'opérateur LIKE et de caractère « **Jocker** ».
**SELECT num_client FROM client WHERE nom LIKE ‘DU%’;**
Recherche les clients dont le nom commence par les deux lettres et suivi de n'importe quelle suite caractères.
**SELECT * FROM client WHERE nom LIKE ‘D_D%’**
Recherche les clients dont le nom comprend la lettre « D » en premier et en troisième position les autres caractères étant quelconques. Le caractère %est utilisé pour symboliser n'importe quelle suite de zéro, un ou

plusieurs caractères.
L'opérateur **IN** permet de tester l'inclusion d'une liste de valeurs.
**SELECT * FROM client WHERE ville IN ('Paris', 'Londres', 'Marseille') ;**
Cette requête sélectionne les clients dont la ville est incluse dans la liste citée.
La condition inverse peut être testée avec l'opérateur **NOT IN.**
Les opérations de comparaison simples **(=, !=, >=,>, <= et<)** peuvent être complétées par les qualificateur **ANY** ou **ALL**.
**SELECT * FROM client WHERE ville<>ALL ('Paris', 'Londres', 'Marseille');**
Recherché les clients qui ne sont localisés dans aucune des trois villes.
**<>ALL ≈ NOT IN, =ANY≈IN**
**SELECT * FROM client WHERE chiffre_affaire >= ALL (SELECT chiffre_affaire FROM CLIENT);**
Sélectionne-le ou les clients dont le chiffre d'affaire est plus élevé.
Si l'on veut trier selon un certain critère, on ajoute une clause **ORDER BY** qui peut comprendre des noms de colonnes, des expressions calculées sur les colonnes ou des numéros de position de ces colonnes ou expression dans la clause **SELECT.**
On peut préciser le sens du tri à l'aide du qualificatif **ASC** (croissant) ou **DESC** (décroissant).

La requête suivante comptera les enregistrements dans la table Employes

**SELECT count(*) FROM Employes;**

La fonction COUNT est utilisée pour compter le nombre de lignes dans une table de base de données. Il peut fonctionner sur les types de données numériques et non numériques.

La fonction **SUM** renvoie la somme de toutes les valeurs de la colonne spécifiée. **SUM** fonctionne uniquement sur les champs numériques.

La requête suivante renvoie la somme des salaires

**SELECT SUM(Salaire) FROM Employes;**

La fonction **AVG** renvoie la moyenne des valeurs d'une colonne spécifiée. Tout comme la fonction **SUM**, elle ne fon sur les types de données numériques.

La requête suivante renvoie le salaire moyen de la table Employes

**SELECT AVG(Salaire) FROM Employes;**

La fonction **MIN** est utilisée pour déterminer la plus petite valeur de toutes les valeurs sélectionnées d'une colonne.

La requête suivante renvoie le salaire minimum de la table Employes

**SELECT MIN(Salaire) FROM Employes;**

Comme son nom l'indique, la fonction **MAX** est l'opposé de la fonction **MIN**. Elle renvoie la plus grande valeur de toutes les valeurs sélectionnées d'une colonne.

La requête suivante renvoie le salaire maximum de la table Employes

**SELECT MIN(Salaire) FROM Employes;**

## c. Langage de manipulation de données (LMD) :

- Le langage SQL fournit des instructions pour ajouter des nouveaux tuples à une relation. Il offre ainsi une interface standard pour ajouter des informations dans une base de données.

Il existe deux moyens d'ajouter des données, soit par fourniture directe des valeurs des propriétés du tuple à ajouter, soit par sélection des tuples à ajouter depuis une autre relation.

INSERT INTO <Nom de la relation> (<Liste ordonnée des propriétés à valoriser>)

VALUES (<Liste ordonnée des valeurs à affecter aux propriétés spécifiées ci-dessus>)

Remarque :

-Les propriétés non valorisées sont affectées à la valeur NULL.

-Il est possible de ne pas spécifier les propriétés à valoriser, dans ce cas, toutes les propriétés de la relation seront considérées, dans leur ordre de définition dans la relation (à n'utiliser que dans les cas les plus simples).

- Le langage SQL fournit une instruction pour modifier des tuples existants dans une relation.

UPDATE <Nom de la relation>

SET <Liste d'affectations Propriété=Valeur, Propriété=Valeur>

WHERE <Condition pour filtrer les tuples à mettre à jour>

- Le langage SQL fournit une instruction pour supprimer des tuples existants dans une relation.

DELETE FROM <Nom de la relation>

WHERE <Condition pour filtrer les tuples à supprimer>