



# Introduction aux Bases de données

1. Introductions aux SGBD et au modèle relationnel.
2. Langage d'interrogation SQL

# Introduction

- 1960 Uniquement des systèmes de gestion de fichiers plus ou moins sophistiqués
- 1970 Début des SGBD réseaux et hiérarchiques proches des systèmes de gestion de fichiers
- 1980 Les SGBDr font leur apparition sur le marché
- 1990 Les SGBDr dominant le marché et apparaissent les SGBD orientés objets (SGBDOO)

# Qu'est-ce qu'une base de données ?

De façon informelle, on peut considérer une **Base de Données** comme un ensemble structuré de données, centralisées ou non, servant pour les besoins d'une ou plusieurs applications, interrogeables et modifiables par un groupe d'utilisateurs en un temps opportun.

Plus formellement, une BD est un ensemble d'informations exhaustives, non redondantes, structurées et persistantes, concernant un sujet.

# Système de Gestion de Base de Données

Un *Système de Gestion de Base de Données* peut être défini comme un ensemble de logiciels prenant en charge la structuration, le stockage, la mise à jour et la maintenance des données. Autrement dit, il permet de décrire, modifier, interroger et administrer les données. C'est, en fait, l'interface entre la base de données et les utilisateurs (qui ne sont pas forcément informaticiens).

## Objectifs d'un SGBD

Un SGBD doit résoudre certains problèmes et répondre à des besoins précis :

- ***Indépendance physique*** : la façon de définir les données doit être indépendante des structures utilisées pour leur stockage
- ***Indépendance logique*** : un utilisateur doit pouvoir percevoir seulement la partie des données qui l'intéresse (c'est ce que l'on appelle une *vue*) et modifier la structure de celle-ci sans remettre en cause la majorité des applications
- ***Manipulation aisée*** des données par des non informaticiens, ce qui suppose des langages "naturels"
- ***Accès efficaces aux données*** et obtention de résultats aux interrogations en un temps "acceptable"

# Objectifs d'un SGBD

- **Administration centralisée** des données pour faciliter l'évolution de leur structure
  - **Non-redondance** : chaque donnée ne doit être présente qu'une seule fois dans la base afin d'éviter les problèmes lors des mises à jour
  - Cohérence (ou intégrité)** : les données ne doivent présenter ni ambiguïté, ni incohérence, pour pouvoir délivrer sans erreur les informations désirées. Cela suppose un mécanisme de vérification lors de l'insertion, de la modification ou de la suppression de données
  - **Partage** des données pour un accès multi-utilisateur simultané aux mêmes données. Il faut entre autre assurer un résultat d'interrogation cohérent pour un utilisateur consultant une base pendant qu'un autre la modifie
  - **Sécurité** des données : robustesse vis-à-vis des pannes (il faut pouvoir retrouver une base "saine" en cas de plantage au cours de modifications) et protection par des droits contre les accès non autorisés

# Propriétés d'un SGBDr

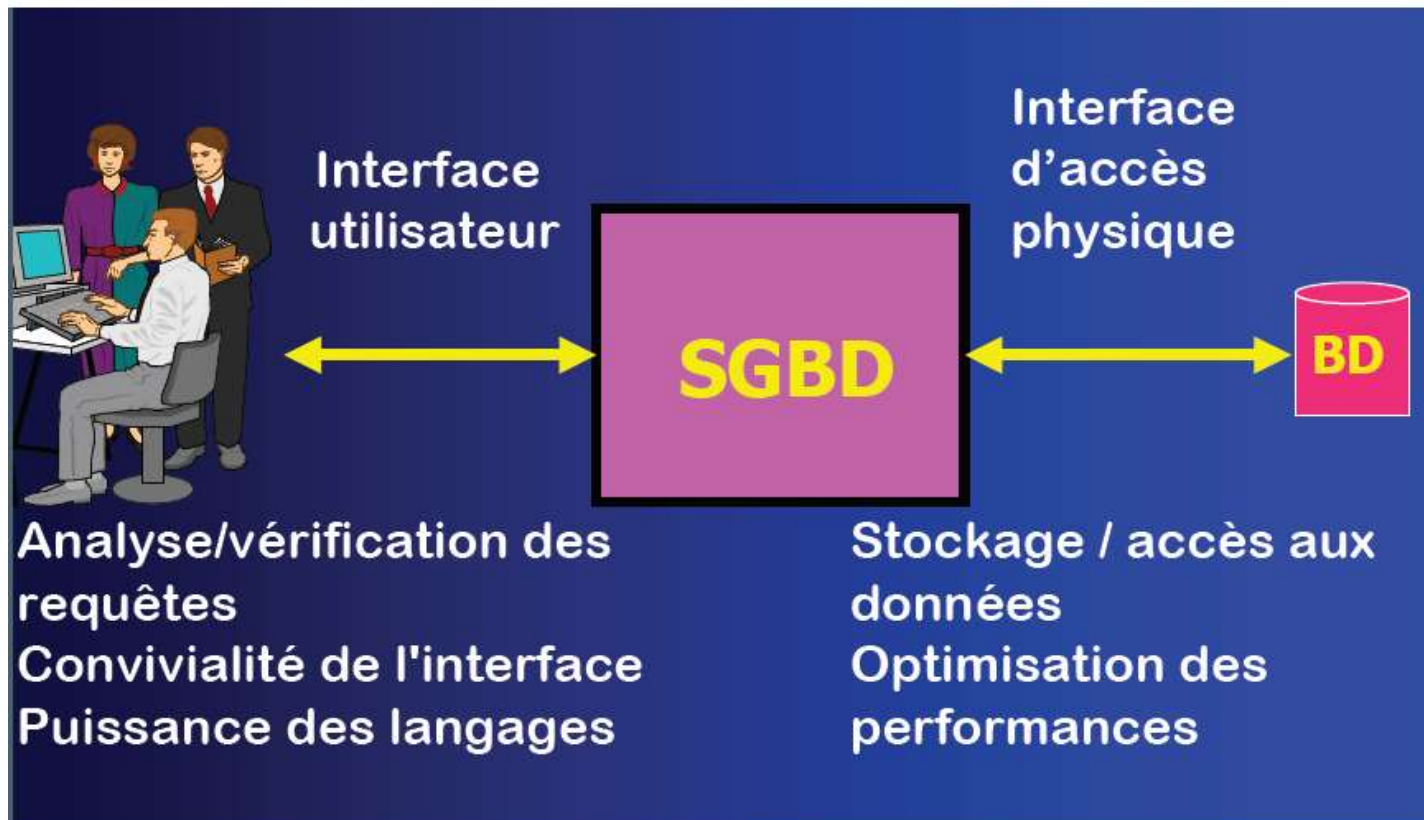
- les propriétés fondamentales d'un SGBDr sont:
  - **Base formelle** reposant sur des principes parfaitement définis
  - **Organisation structurée** des données dans des tables interconnectées (d'où le qualificatif *relationnelles*), pour pouvoir détecter les dépendances et redondances des informations
  - Implémentation d'un **langage relationnel ensembliste** permettant à l'utilisateur de décrire aisément les interrogations et manipulation qu'il souhaite effectuer sur les données
  - **Indépendance des données** vis-à-vis des programmes applicatifs (dissociation entre la partie "stockage de données" et la partie "gestion" - ou "manipulation")
  - **Gestion des opérations concurrentes** pour permettre un accès multi-utilisateur sans conflit
  - **Gestion de l'intégrité des données**, de leur protection contre les pannes et les accès illicites



# Composants des SGBD

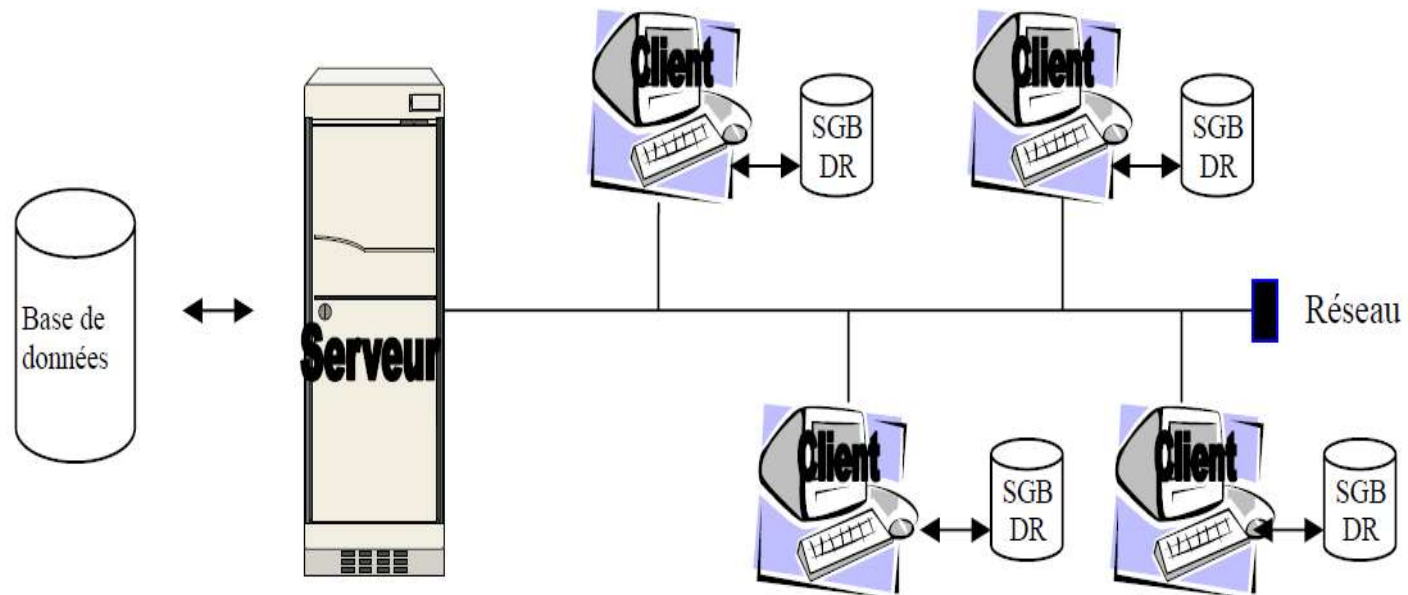
- ***la description des données*** au moyen d'un *Langage de Définition de Données* (LDD). Le résultat de la compilation est un ensemble de tables, stockées dans un fichier spécial appelé dictionnaire (ou répertoire) des données
- ***la manipulation des données*** au moyen d'un *Langage de Manipulation de Données* (LMD) prenant en charge leur consultation et leur modification de façon optimisée, ainsi que les aspects de sécurité
- ***la sauvegarde et la récupération après pannes***, ainsi que des mécanismes permettant de pouvoir revenir à l'état antérieur de la base tant qu'une modification n'est pas finie (notion de *transaction*)
- ***les accès concurrents aux données*** en minimisant l'attente des utilisateurs et en garantissant l'obtention de données cohérentes en cas de mises à jours simultanées

# Architecture d'un SGBD



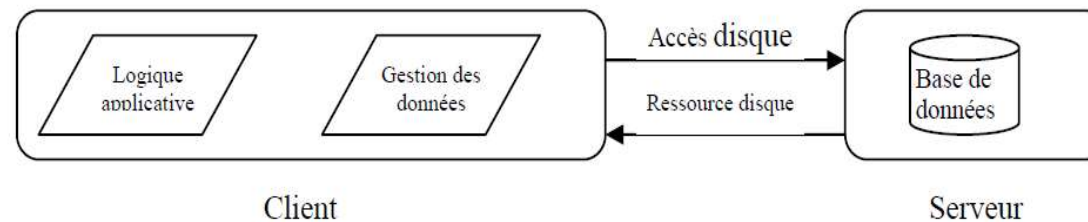
# ARCHITECTURE DES SGBD

## L'architecture à serveur de fichiers



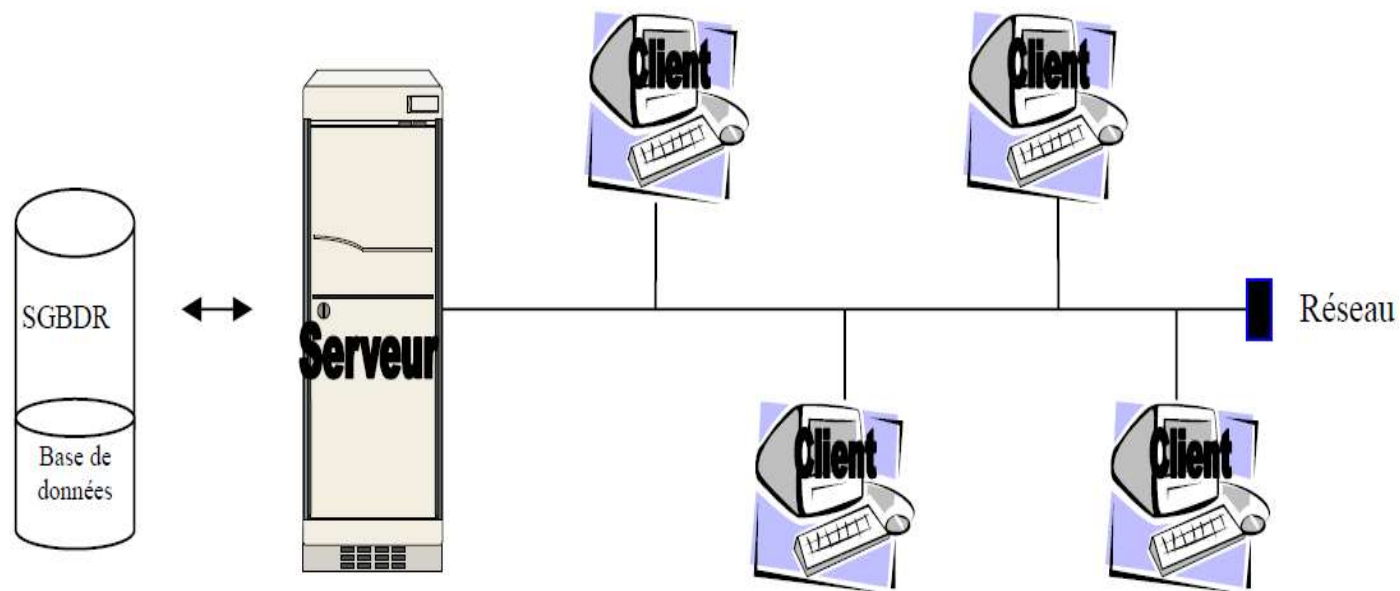
# L'architecture à serveur de fichiers (2)

Le serveur de fichiers héberge l'ensemble des données, mais n'effectue aucun traitement. C'est une fonction du client, tout comme la logique applicative.



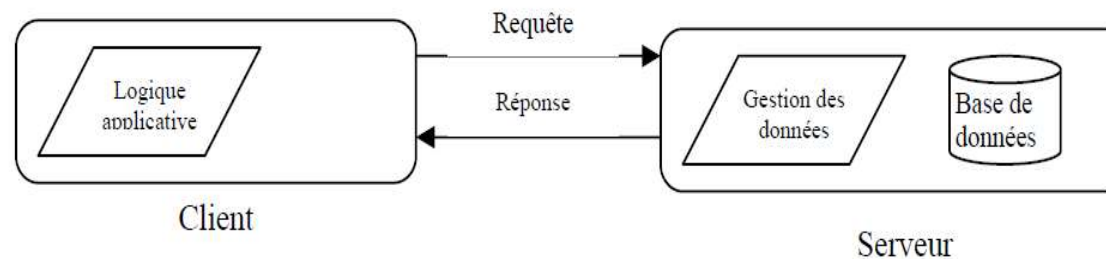
Avantages	Inconvénients
<ul style="list-style-type: none"><li>- Coût faible.</li><li>- Développement d'applications avec peu de compétences.</li><li>- Grande souplesse de l'interface utilisateur.</li></ul>	<ul style="list-style-type: none"><li>- Inadaptés dans le cas d'un grand nombre d'utilisateurs connectés</li><li>- La mise à jour des applications est lourde (les applications sont en effet à mettre à jour au niveau de chaque client)</li><li>- Sécurité, performance et fiabilité faibles.</li></ul>

## L'architecture à serveur de bases de données (1)



## L'architecture à serveur de bases de données (2)

Le serveur prend en charge non seulement l'accès aux données mais aussi l'intégralité des traitements. Le client est dit « léger » car il ne comporte que l'interface de présentation des données (logique applicative).



Pour alléger la charge du serveur, il est possible de répartir la charge de travail sur deux machines différentes :

- une machine « serveur d'applications »
- une machine « serveur de bases de données »

Il est également possible de répartir les traitements.

## L'architecture à serveur de bases de données (3)

Avantages	Inconvénients
<ul style="list-style-type: none"><li>- Sécurité et fiabilité élevées.</li><li>- Hébergement de base de données de très grande taille (milliers de Go).</li><li>- Souplesse de l'interface utilisateur.</li><li>- Gestion centralisée performante.</li></ul>	<ul style="list-style-type: none"><li>- Coût élevé</li><li>- Risque de forte sollicitation du serveur (échanges nombreux)</li><li>- Le serveur doit être surdimensionné (en mémoire vive pour l'exécution des programmes et en disque pour les données). Dans les grandes organisations, le serveur peut être multiprocesseur</li></ul>

Plus précisément, la communication est organisée en couches , garantissant l'indépendance de la solution client par rapport à la solution serveur.

# L'ARCHITECTURE À TROIS NIVEAUX

une architecture intégrant les trois niveaux de schémas : externe, conceptuel et interne

cette architecture permet de bien comprendre les niveaux de description et transformation de données possible dans un SGBD

L'architecture est articulée autour du dictionnaire de données et comporte deux parties :

1. un ensemble de modules (appelés processeurs) permettant d'assurer la description de données et donc la constitution du dictionnaire de données
2. une partie permettant d'assurer la manipulation des données, c'est-à-dire l'interrogation et la mise à jour des bases.



# Trois couches

- Couche externe

description et manipulation des données dédiés a un groupe d'utilisateurs

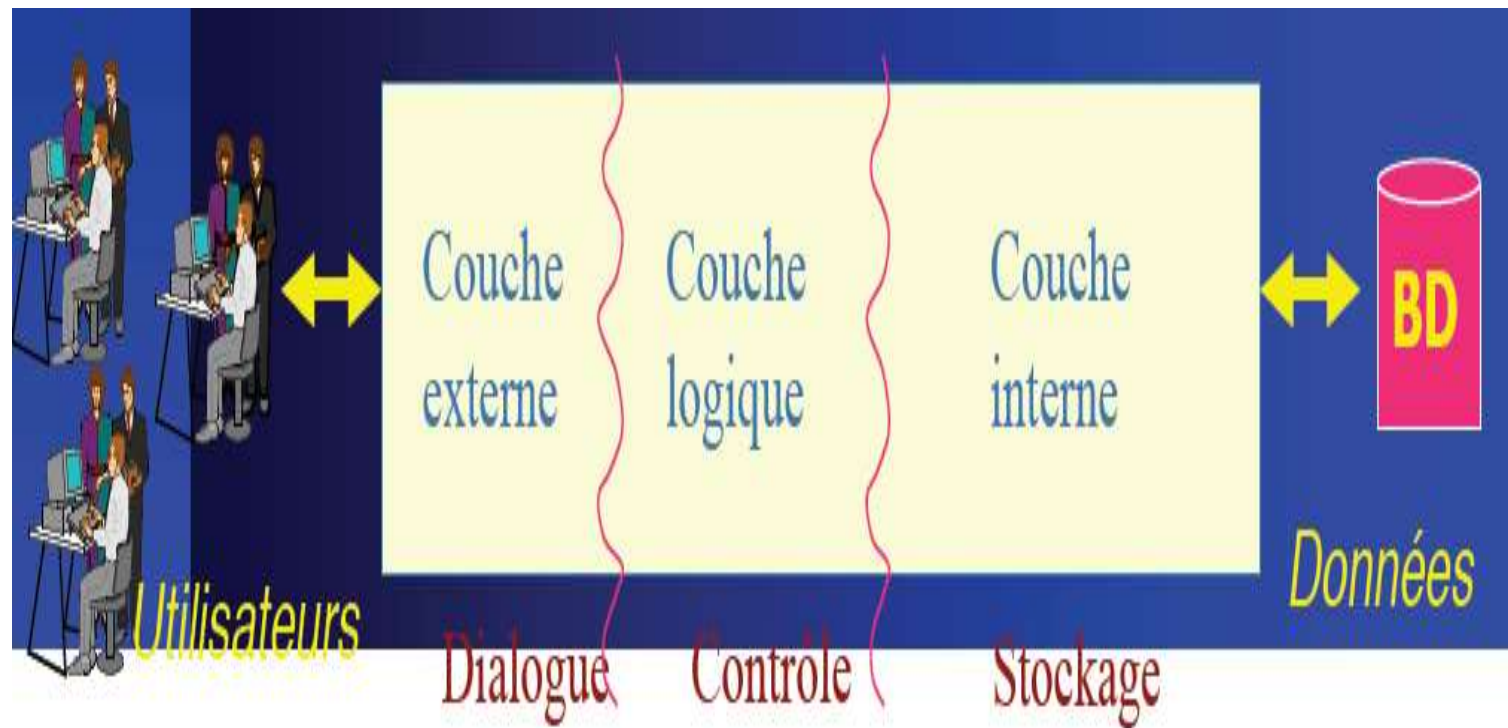
Couche interne

stockage des données sur des supports physiques, gestion des structures de  
mémorisation (fichiers) et d'accès (gestion des index, des clés, ...)

- Couche logique

description et manipulation abstraites des données

## Trois couches (suite)



# Niveau conceptuel (1)

- le schéma conceptuel. Celui-ci peut donc être considéré comme la description du contenu de la base : c'est le résultat d'un travail d'analyse et de conception d'un système d'information automatisé.
- Un schéma conceptuel doit offrir les caractéristiques suivantes :
  1. **puissance de représentation** : aspects structurels, contraintes existant dans l'univers réel.
  2. **stabilité et flexibilité** : l'ajout d'une nouvelle donnée ou d'une nouvelle contrainte ne doit pas entraîner de changement important dans le schéma.
  3. **simplicité de compréhension** : nombre d'éléments réduit, dissociation claire des différents concepts.
  4. **simplicité d'utilisation** : nombre restreint d'outils ou de primitives de manipulation.
  5. **base formelle** : la définition du schéma doit s'appuyer sur une méthode rigoureuse, mathématique, pour éviter toute ambiguïté d'interprétation et pour garantir la fiabilité des données.

## Niveau externe

- Le niveau externe comprend les "vues" spécifiques définies pour la manipulation des données. Il prend en compte les contraintes d'accès imposées par la nature des applications à considérer (indépendamment des caractéristiques techniques) et exprime les besoins en données des différents utilisateurs, ou applications.

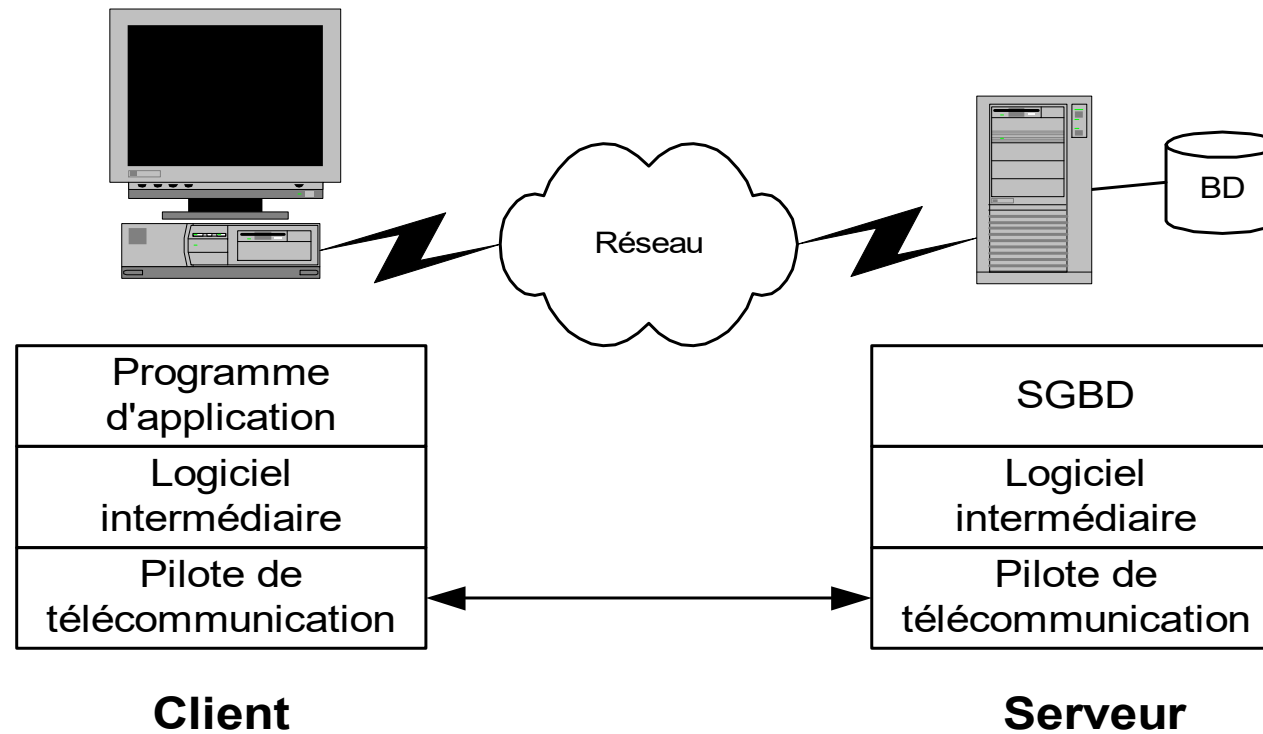
## Niveau interne ou Physique

- Il correspond à la représentation en machine, aussi efficace que possible, du schéma conceptuel : le schéma physique intègre les caractéristiques techniques (choix du SGBD, du matériel, du système d'exploitation...).
- L'efficacité doit tenir compte d'une part des contraintes d'implantation (taille des disques, optimisation du système de fichiers...), d'autre part des critères d'utilisation (traitement interactif ou en batch, selon la fréquence d'utilisation et la durée du traitement...).

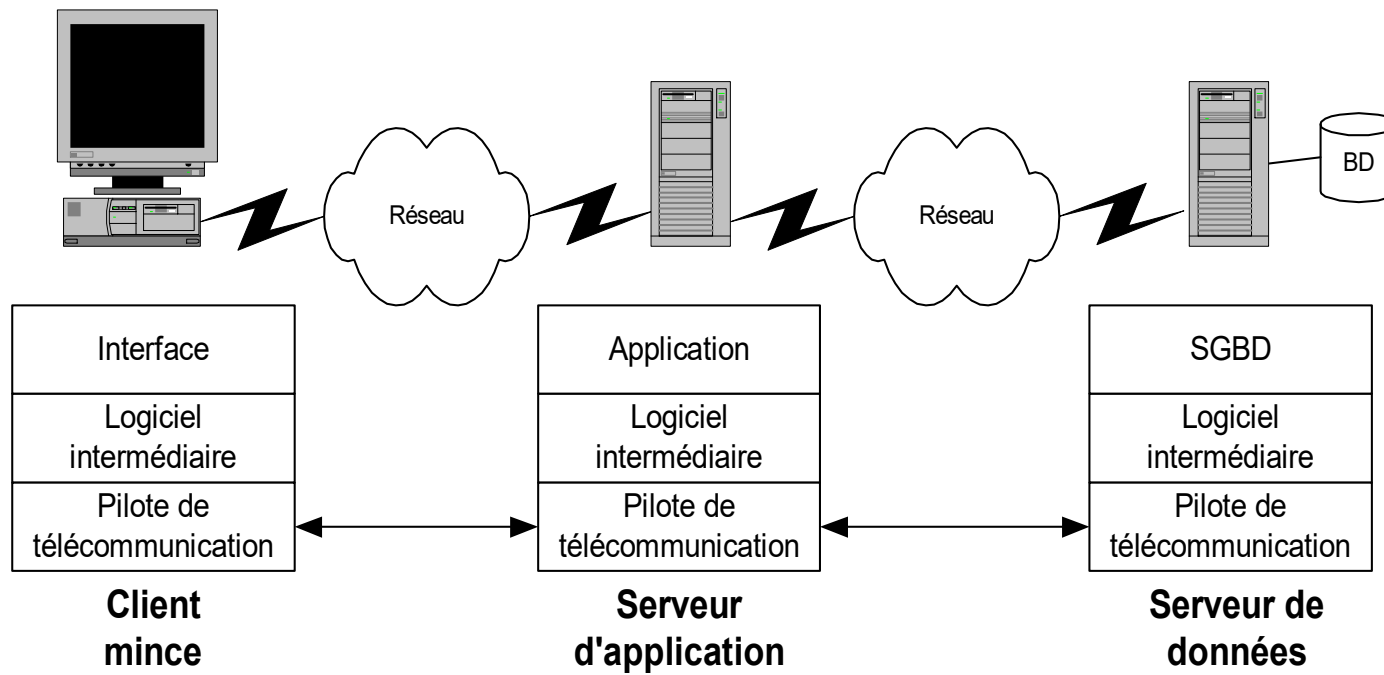
## Architecture du type client-serveur (client-server architecture)

- programme d'application = *client*
  - interface (« GUI ») + traitement du domaine d 'application
- SGBD = *serveur de données* « *data server* »

# Architecture client / serveur



# Architecture 3 tiers



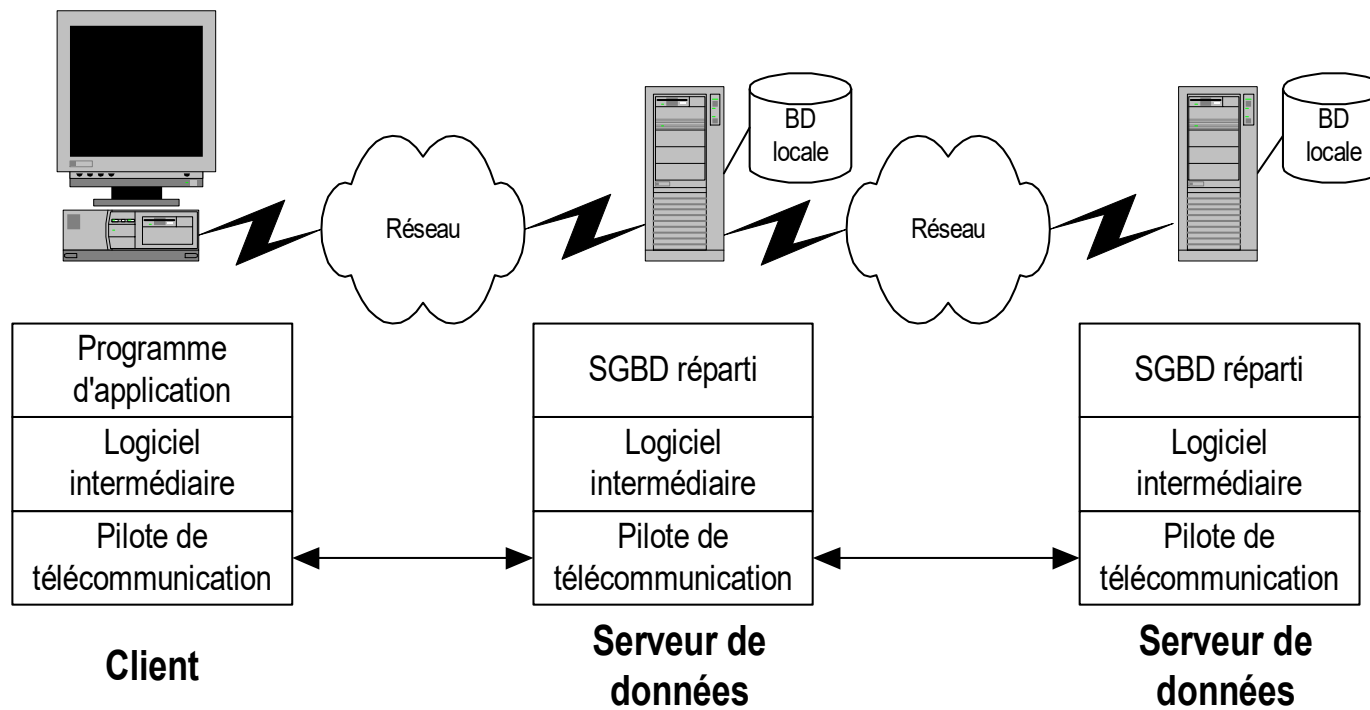


# Base de données distribuées(1)

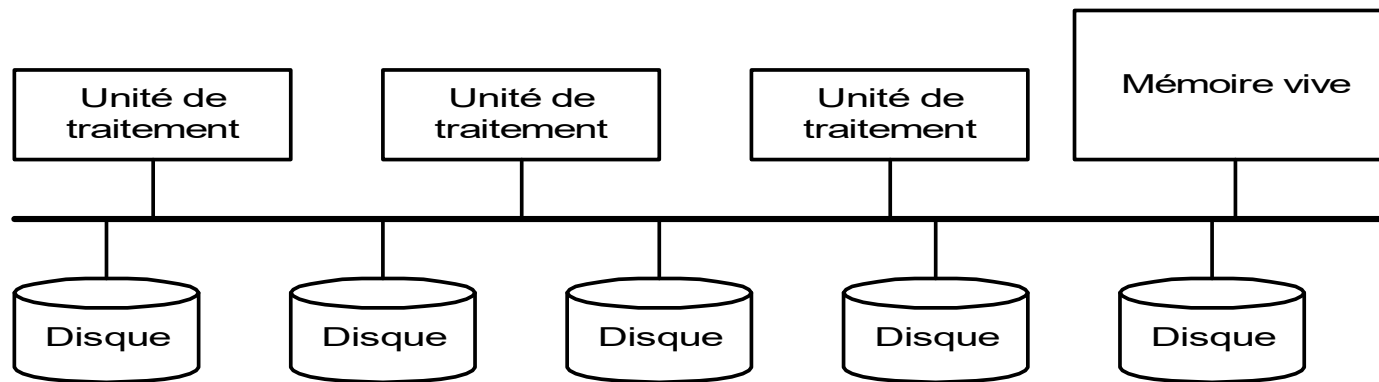
BD réparties:

- Les données sont distribuées et/ou dupliquées sur différents sites du réseau (ex: internet) qui possèdent un certain degré d'autonomie. Chaque site peut comporter une BD parallèle

## Base de données distribuées(2)



## Base de données parallèles



BD parallèles: Les données peuvent être distribuées sur plusieurs disques d'un même site, et l'exécution des requêtes peut être parallélisée sur les différentes unités de traitement (CPU) du site.

# Entrepôt de données

- *Base de données opérationnelle*
  - traitement des données quotidiennes et récentes
- *Entrepôt de données (data wharehouse)*
  - grand volume de données historiques extraites de bases opérationnelles pour le support à la prise de décision

# LES TRANSACTIONS

Un utilisateur ou un programme d'application interagit avec la base de données via des appels au SGBD exprimés sous forme de requêtes SQL. Le SGBD garantit une *qualité de service* absolue dans l'exécution de ces requêtes, garantie qui comporte quatre propriétés : *atomicité*, *cohérence*, *isolation* et *durabilité* (ou ACID).

# LES TRANSACTIONS

**Atomicité:** L'exécution d'une requête est atomique si cette dernière, quoi qu'il arrive, est exécutée complètement (en cas de réussite) ou pas du tout (en cas d'échec ou d'incident).

**Cohérence.** Une requête de modification exécutée sur des données cohérentes laisse celles-ci dans un état final également cohérent. Concrètement, le SGBD garantit le respect de toutes les contraintes d'intégrité imposées aux données.

**Isolation.** Les opérations sur les données sont exécutées comme si chaque requête disposait de la base de données pour elle seule.

**Durabilité.** Lorsque le SGBD a confirmé qu'une mise à jour a été effectuée avec succès, il garantit qu'elle est permanente.

# Organisation de la mise en oeuvre d 'un SGBD

Quatre catégories de fonctions sont impliquées dans cette *gestion de données* :

- les tâches liées à l'***architecture de données*** consistent à analyser, classifier et structurer les données au moyen de modèles confirmés
- l'***administration de données*** vise à superviser l'adéquation des définitions et des formats de données avec les directives de standardisation et les normes internationales, à conseiller les développeurs et les utilisateurs, et à s'assurer de la disponibilité des données à l'ensemble des applications. L'administrateur assume en outre des responsabilités importantes dans la maintenance et la gestion des autorisations d'accès
- les professionnels en ***technologie de données*** ont en charge l'installation, la supervision, la réorganisation, la restauration et la protection des bases. Ils en assurent aussi l'évolution au fur et à mesure des progrès technologiques dans ce domaine.
- l'***exploitation de données*** consiste à mettre à disposition des utilisateurs les fonctions de requête et de reporting (générateurs d'*états*), ainsi qu'à assurer une assistance aux différents services pour qu'ils puissent gérer leur stock propre de données en autonomie (service *infocentre*).

# CONCEPTS DES BASES DE DONNEES

- Tables, lignes et colonnes

CLIENT					
NCLI	NOM	ADRESSE	LOCALITE	(CAT)	COMPTE
B062	GOFFIN	72, r. de la Gare	Namur	B2	-3200
B112	HANSENNE	23, r. Dumont	Poitiers	C1	1250
B332	MONTI	112, r. Neuve	Genève	B2	0
B512	GILLET	14, r. de l'Eté	Toulouse	B1	-8700
C003	AVRON	8, r. de la Cure	Toulouse	B1	-1700
C123	MERCIER	25, r. Lemaître	Namur	C1	-2300
C400	FERARD	66, r. du Tertre	Poitiers	B2	350
D063	MERCIER	201, bvd du Nord	Toulouse		-2250
F010	TOUSSAINT	5, r. Godefroid	Poitiers	C1	0
F011	PONCELET	17, Clos des Erables	Toulouse	B2	0
F400	JACOB	78, ch. du Moulin	Bruxelles	C2	0
K111	VANBIST	180, r. Florimont	Lille	B1	720
K729	NEUMAN	40, r. Bransart	Toulouse		0
L422	FRANCK	60, r. de Wépion	Namur	C1	0
S127	VANDERKA	3, av. des Roses	Namur	C1	-4580
S712	GUILLAUME	14a, ch. des Roses	Paris	B1	0



# CONCEPTS DES BASES DE DONNEES

- Tables, lignes et colonnes

CLIENT						schéma
NCLI	NOM	ADRESSE	LOCALITE	(CAT)	COMPTE	
B062	GOFFIN	72, r. de la Gare	Namur	B2	-3200	données
B112	HANSENNE	23, r. Dumont	Poitiers	C1	1250	
B332	MONTI	112, r. Neuve	Genève	B2	0	
B512	GILLET	14, r. de l'Été	Toulouse	B1	-8700	
C003	AVRON	8, r. de la Cure	Toulouse	B1	-1700	
C123	MERCIER	25, r. Lemaître	Namur	C1	-2300	
C400	FERARD	65, r. du Tertre	Poitiers	B2	350	
D063	MERCIER	201, bvd du Nord	Toulouse		-2250	
F010	TOUSSAINT	5, r. Godefroid	Poitiers	C1	0	
F011	PONCELET	17, Clos des Erables	Toulouse	B2	0	
F400	JACOB	78, ch. du Moulin	Bruxelles	C2	0	
K111	VANBIST	180, r. Florimont	Lille	B1	720	
K729	NEUMAN	40, r. Bransart	Toulouse		0	
L422	FRANCK	60, r. de Wépion	Namur	C1	0	
S127	VANDERKA	3, av. des Roses	Namur	C1	-4560	
S712	GUILLAUME	14a, ch. des Roses	Paris	B1	0	

ligne

colonne obligatoire

colonne facultative

## Valeur *null*

- L'absence de valeur est indiquée par un marqueur spécial, dit **valeur null**. Généralement représenté par <null> ou par rien.

### Problème : plusieurs interprétations possibles

1. information pertinente mais inexistante pour l'entité
  2. information non pertinente pour cette entité
  3. information existante mais actuellement inconnue
- **Recommandation : éviter si possible les colonnes facultatives.**

# Identifiants et clés étrangères

- Un **identifiant** est un groupe de colonnes d'une table T tel qu'il ne puisse, à tout moment, exister plus d'une ligne dans T qui possède des valeurs déterminées pour ces colonnes. La valeur de l'identifiant permet de désigner une ligne de T.

Une **clé étrangère** identifie une colonne ou un ensemble de colonnes d'une table comme référençant une colonne ou un ensemble de colonnes d'une autre table (la table référencée)

# Identifiants et clés étrangères



# Identifiants et clés étrangères

- Un **identifiant** définit une **contrainte d'unicité**. Il existe d'autres moyens de définir cette contrainte.
- Une table peut posséder plusieurs identifiants. On choisit l'un d'eux, qu'on déclare **primaire**. Les autres sont dès lors **secondaires**
- L'identifiant primaire est constitué de **colonnes obligatoires**
- Un identifiant est **minimal** si chacune de ses colonnes est nécessaire pour garantir la contrainte d'unicité.
- Il est possible de déclarer une **table sans identifiant** mais ceci n'est pas recommandé.

# Identifiants et clés étrangères

- Une **clé étrangère** définit une **contrainte référentielle**. Il existe d'autres moyens de définir cette contrainte.
- Si une des colonnes d'une clé étrangère est facultative, il est recommandé de les rendre **toutes facultatives**.
- Une clé étrangère référence en principe l'**identifiant primaire** de la table cible.
- Une clé étrangère et l'identifiant qu'elle référence ont la **même composition** : même nombre de colonnes et colonnes de mêmes types prises deux à deux.
- Il se peut qu'une clé étrangère soit également un identifiant.
- Il se peut que les colonnes d'une clé étrangère appartiennent, en tout ou en partie, à un identifiant.

# Identifiants et clés étrangères

- Un **identifiant** minimal est aussi appelé **clé candidate** (*candidate key*). [\*]
- Un **identifiant primaire** s'appelle aussi **clé primaire** (*primary key*).
- Il n'existe pas d'autre terme pour désigner les **identifiants secondaires**
- **Clé étrangère** = *foreign key*.

## Schéma et contenu

CLIENT					
NCLI	NOM	ADRESSE	LOCALITE	(CAT)	COMPTE
B062	GOFFIN	72, r. de la Gare	Namur	B2	-3200
B112	HANSENNE	23, r. Dumont	Poitiers	C1	1250
B332	MONTI	112, r. Neuve	Genève	B2	0
B512	GILLET	14, r. de l'Eté	Toulouse	B1	-8700
C003	AVRON	8, r. de la Cure	Toulouse	B1	-1700
C123	MERCIER	25, r. Lemaitre	Namur	C1	-2300
C400	FERARD	65, r. du Tertre	Poitiers	B2	350
D063	MERCIER	201, bvd du Nord	Toulouse		-2250
F010	TOUSSAINT	5, r. Godefroid	Poitiers	C1	0
F011	PONCELET	17, Clos des Erables	Toulouse	B2	0
F400	JACOB	78, ch. du Moulin	Bruxelles	C2	0
K111	VANBIST	180, r. Florimont	Lille	B1	720
K729	NEUMAN	40, r. Bransart	Toulouse		0
L422	FRANCK	60, r. de Wépion	Namur	C1	0
S127	VANDERKA	3, av. des Roses	Namur	C1	-4580
S712	GUILLAUME	14a, ch. des Roses	Paris	B1	0

*le schéma*

*les données*



# Schéma et contenu

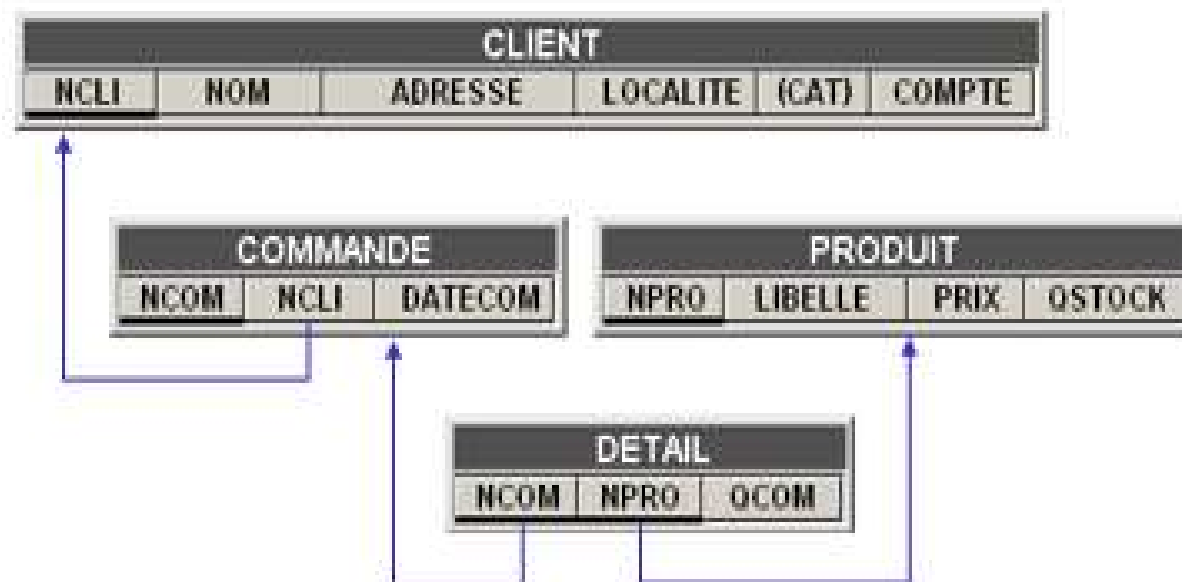
Le **schéma d'une table** définit sa structure. Il spécifie notamment :

1. le nom de la table,
2. pour chaque colonne, son nom, son type, son caractère obligatoire,
3. l'identifiant primaire (liste des colonnes)
4. les identifiants secondaires éventuels (liste des colonnes)
5. les clés étrangères éventuelles (liste des colonnes et table cible).

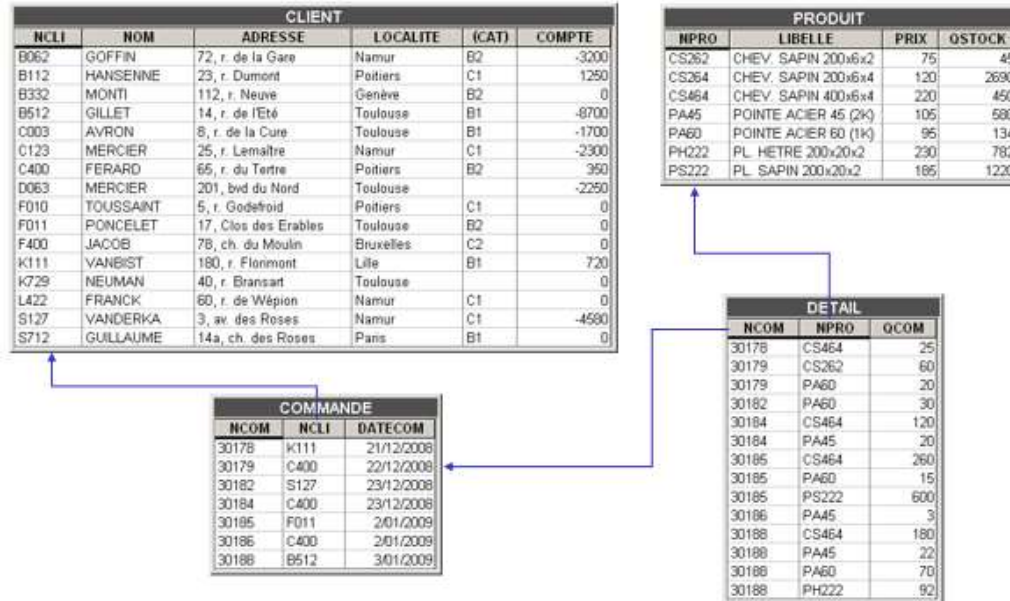
Le **contenu d'une table** est formé d'un ensemble de lignes conformes au schéma.

- Le **contenu d'une table** est sujet à de fréquentes modifications. Le **schéma d'une table** peut évoluer mais moins fréquemment

## Exemple de base de données

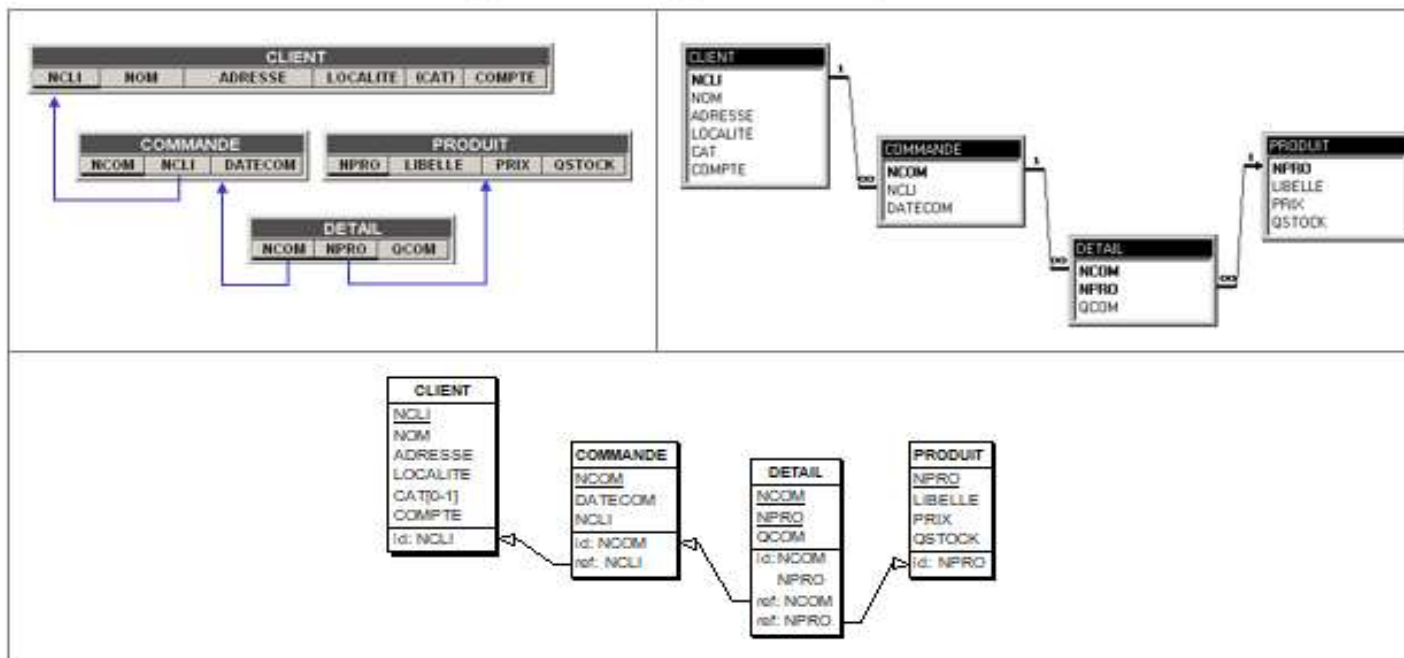


# Exemple de base de données



# Exemple de base de données

## Variantes de schéma



# Modifications et contraintes d'intégrité

Une **contrainte d'intégrité** est une règle qui définit la cohérence d'une donnée ou d'un ensemble de données de la BD. Le modèle relationnel impose les contraintes structurelles suivantes :

**INTÉGRITÉ DE DOMAINE**

**INTÉGRITÉ DE CLÉ**

**INTÉGRITÉ RÉFÉRENCIELLE**

Il existe trois opérations élémentaires de modification :

1. insérer une ligne
2. supprimer une ligne
3. modifier une valeur de colonne d'une ligne.

# INTÉGRITÉ DE DOMAINE

Les valeurs d'une colonne de relation doivent appartenir au domaine correspondant

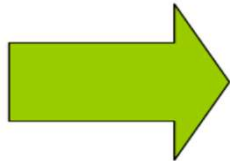


- contrôle des valeurs des attributs
- contrôle entre valeurs des attributs

# INTÉGRITÉ DE CLÉ

Les valeurs de clés primaires doivent être :

- uniques
- non NULL



- Unicité de clé
- Unicité des n-uplets

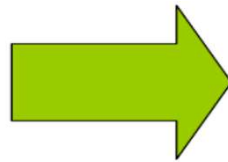
- Valeur NULL

valeur conventionnelle pour représenter une information **inconnue**

- dans toute extension possible d'une relation, il ne peut exister 2 n-uplets ayant même valeur pour les attributs clés sinon 2 clés identiques détermineraient 2 lignes identiques (d'après la définition d'une clé), ce qui est absurde

## INTÉGRITÉ RÉFÉRENTIELLE

Les valeurs de clés étrangères sont 'NULL' ou sont des valeurs de la clé primaire auxquelles elles font référence



- Relations dépendantes

- LES DÉPENDANCES :

Liaisons de un à plusieurs exprimées par des attributs particuliers: **clés étrangères** ou **clés secondaires**



Les contraintes de référence ont un impact important pour les opérations de mises à jour, elles permettent d'éviter les **anomalies** de mises à jour

Exemple :

CLIENT (**no\_client**, nom, adresse)

ACHAT (**no\_produit**, no\_client, date, qte)

Clé étrangère no\_client dans ACHAT

- **insertion** tuple no\_client = X dans ACHAT
  - ⇒ vérification si X existe dans CLIENT
- **suppression** tuple no\_client = X dans CLIENT
  - ⇒ soit interdire si X existe dans ACHAT
  - ⇒ soit supprimer en cascade tuple X dans ACHAT
  - ⇒ soit modifier en cascade X = NULL dans ACHAT
- **modification** tuple no\_client = X en X' dans CLIENT
  - ⇒ soit interdire si X existe dans ACHAT
  - ⇒ soit modifier en cascade X en X' dans ACHAT

## Redondances internes

**Table répertoriant les livres d'une bibliothèque :**

LIVRE					
NUMERO	TITRE	AUTEUR	ISBN	DATE_ACHAT	EMPL
1029	L'humanité perdue	Finkelkraut A.	2 02 033300 7	14/10/2008	F3
1030	L'humanité perdue	Finkelkraut A.	2 02 033300 7	14/10/2008	F3
1032	Mercure	Nothomb A.	2 253 14911 X	14/10/2008	G5
1045	Eva Luna	Allende I.	2 253 05354 6	22/2/2009	F3
1067	Mercure	Nothomb A.	2 253 14911 X	24/2/2009	G5
1022	Mercure	Nothomb A.	2 253 14911 X	3/10/2008	G6

## Redondances internes

### Observation

Les données TITRE et AUTEUR sont répétées autant de fois qu'il existe de livres identiques.

LIVRE					
NUMERO	TITRE	AUTEUR	ISBN	DATE_ACHAT	EMPL
1029	L'humanité perdue	Finkelkraut A.	2 02 033300 7	14/10/2008	F3
1030	L'humanité perdue	Finkelkraut A.	2 02 033300 7	14/10/2008	F3
1032	Mercure	Nothomb A.	2 253 14911 X	14/10/2008	G5
1045	Eva Luna	Allende I.	2 253 05354 6	22/2/2009	F3
1067	Mercure	Nothomb A.	2 253 14911 X	24/2/2009	G5
1022	Mercure	Nothomb A.	2 253 14911 X	3/10/2008	G6

Cette table viole le principe premier des bases de données : ***tout fait du domaine d'application est enregistré une et une seule fois.***

## Redondances internes

LIVRE					
NUMERO	TITRE	AUTEUR	ISBN	DATE_ACHAT	EMPL
1029	L'humanité perdue	Finkelkraut A.	2 02 033300 7	14/10/2008	F3
1030	L'humanité perdue	Finkelkraut A.	2 02 033300 7	14/10/2008	F3
1032	Mercure	Nothomb A.	2 253 14911 X	14/10/2008	G5
1045	Eva Luna	Allende I.	2 253 05354 6	22/2/2009	F3
1067	Mercure	Nothomb A.	2 253 14911 X	24/2/2009	G5
1022	Mercure	Nothomb A.	2 253 14911 X	3/10/2008	G6

### Problèmes

- gaspillage d'espace
- si on modifie la valeur d'un titre, il faut répercuter cette modification dans toutes les lignes similaires
- si on supprime l'unique exemplaire d'un livre, on perd les informations sur son auteur et son titre

# Redondances internes

## Suggestion

Rassembler les données communes (ISBN, TITRE, AUTEUR) dans une table spécifique

OUVRAGE		
ISBN	TITRE	AUTEUR
2 02 033300 7	L'humanité perdue	Finkelkraut A.
2 253 14911 X	Mercure	Nothomb A.
2 253 05354 6	Eva Luna	Allende I.

EXEMPLAIRE			
NUMERO	ISBN	DATE_ACHAT	EMPL
1029	2 02 033300 7	14/10/2008	F3
1030	2 02 033300 7	14/10/2008	F3
1032	2 253 14911 X	14/10/2008	G5
1045	2 253 05354 6	22/2/2009	F3
1067	2 253 14911 X	24/2/2009	G5
1022	2 253 14911 X	3/10/2008	G6

# Redondances internes

## Deux questions

1. Comment détecter les situations de redondance ?
2. Comment les corriger ?

**La réponse à ces questions repose sur une nouvelle forme de contrainte d'intégrité : *la dépendance fonctionnelle*.**

# LES DÉPENDANCES FONCTIONNELLES

## Dépendance fonctionnelle

- Soit  $R(A_1, A_2, \dots, A_n)$  un schéma de relation

Soit  $X$  et  $Y$  des sous ensembles de  $\{A_1, A_2, \dots, A_n\}$

On dit que  $Y$  dépend fonctionnellement de  $X$  ( $X \rightarrow Y$ ) si à chaque valeur de  $X$  correspond une valeur unique de  $Y$

- on écrit :  $X \rightarrow Y$
- on dit que :  $X$  détermine  $Y$
- Ex.:

$$\text{no\_client} \rightarrow \text{nom\_client}$$

Un nom du client ne peut pas avoir deux numéros.

$$\text{no\_CIN} \rightarrow \text{nom\_client}$$

On ne peut pas avoir deux citoyens qui ont le même no-CIN

La réciproque est fausse

# LES DÉPENDANCES FONCTIONNELLES

- La D.F. peut porter sur la concaténation de plusieurs propriétés.
- $\text{Num\_Commande} + \text{nom\_Client} \stackrel{\text{df}}{\rightarrow} \text{quantité\_commande}$

- **D.F. élémentaire**

On dit qu'il ya une D.F. élémentaire entre A et B, si  $A \rightarrow B$  et si aucune partie de A ne détermine B (l'ensemble OK)

- **Exemple:**

$\text{Numero Client} + \text{Nom Client} \stackrel{\text{df}}{\rightarrow} \text{Adresse}$  (ne vérifie pas la condition)

$\text{Numero Client} \rightarrow \text{Adresse}$  (OK)

- **D.F.E directe**

- Si cette dépendance est une dépendance élémentaire  $A \rightarrow B$  et s'il n'existe pas de propriété C tel que  $A \rightarrow C$  et  $C \rightarrow B$  ( on élimine la transitivité).

$\text{Num\_Cmd} \rightarrow \text{Num\_Clt}$  (directe)

$\text{Num\_Clt} \rightarrow \text{Nom\_Clt}$  (directe)

$\text{Num\_Cmd} \rightarrow \text{Nom\_Clt}$  (indirecte)



## Redondances internes

- **Notion de dépendance fonctionnelle**

LIVRE					
NUMERO	TITRE	AUTEUR	ISBN	DATE_ACHAT	EMPL
1029	L'humanité perdue	Finkelkraut A.	2 02 033300 7	14/10/2008	F3
1030	L'humanité perdue	Finkelkraut A.	2 02 033300 7	14/10/2008	F3
1032	Mercure	Nothomb A.	2 253 14911 X	14/10/2008	G5
1045	Eva Luna	Allende I.	2 253 05354 6	22/2/2009	F3
1067	Mercure	Nothomb A.	2 253 14911 X	24/2/2009	G5
1022	Mercure	Nothomb A.	2 253 14911 X	3/10/2008	G6

**ISBN → TITRE, AUTEUR**

**si deux lignes ont la même valeur de ISBN,  
alors elles ont aussi les mêmes valeurs de TITRE et d'AUTEUR**

**On dit que :**

- il existe une dépendance fonctionnelle de ISBN vers TITRE et AUTEUR
- ISBN *détermine* ou *est un déterminant* de TITRE et AUTEUR
- TITRE et AUTEUR *dépendent de* ou *sont déterminés par* ISBN

# Redondances internes

## Notion de dépendance fonctionnelle

### Deux observations

1. par définition, un identifiant détermine toutes les colonnes de la table
2. si un groupe de colonnes détermine chaque colonne de la table, il constitue par définition un identifiant de la table

**NUMERO  $\longrightarrow$  TITRE, AUTEUR, ISBN, DATE\_ACHAT, EMPL**

# Redondances internes

Comment détecter les situations de redondance ?

## **Réponse**

Il y a redondance interne dès qu'il existe un déterminant qui n'est pas un identifiant de la table

Une dépendance fonctionnelle dont le déterminant n'est pas un identifiant est dite **anormale**

*ISBN est un déterminant dans LIVRE mais il n'en est pas un identifiant. Il entraîne donc des redondances internes.*

# Redondances internes

## Comment corriger les situations de redondance

### Réponse

En décomposant la table T en deux fragments T1 et T2 :

T1(déterminant, déterminé)

T2.déterminant est une *clé étrangère* vers T1

OUVRAGE		
ISBN	TITRE	AUTEUR
2 02 033300 7	L'humanité perdue	Finkelkraut A.
2 253 14911 X	Mercure	Nothomb A.
2 253 05354 6	Eva Luna	Allende I.

EXEMPLAIRE			
NUMERO	ISBN	DATE_ACHAT	EMPL
1029	2 02 033300 7	14/10/2008	F3
1030	2 02 033300 7	14/10/2008	F3
1032	2 253 14911 X	14/10/2008	G5
1045	2 253 05354 6	22/2/2009	F3
1067	2 253 14911 X	24/2/2009	G5
1022	2 253 14911 X	3/10/2008	G6

# Redondances

**Synonyme** : termes différents qui désignent le même concept du domaine d'application.

*Exemples* : **assuré** et **client** dans un compagnie d'assurance

**Homonyme** : termes identiques, mais qui désignent des concepts différents dans certaines parties du domaine d'application.

*Exemples* : **adresse** de commande, **adresse** d'expédition, **adresse** de facturation; **unité** de fabrication, **unité** de soins

# Redondances internes

## Dernières remarques

1. Une table qui est le siège d'une dépendance fonctionnelle anormale est dite **non normalisée**
2. Une table sans dépendance fonctionnelle anormale est dite **normalisée**
3. Décomposer une table de manière à éliminer ses dépendances anormales consiste à **normaliser cette table**
4. Il est essentiel que toutes les tables d'une base de données soient normalisées

# LES FORMES NORMALES

## La théorie de la normalisation

**La théorie de la normalisation** est une **théorie** destinée à concevoir un bon schéma d'une base de données sans redondance d'information et sans risques d'anomalie de mise à jour. Elle a été introduite dès l'origine dans le modèle relationnel. Elles traduisent des contraintes sur les données.

## La décomposition

### Objectif:

- décomposer les relations du schéma relationnel sans perte d'informations
- aboutir au schéma relationnel normalisé
  - Le schéma de départ est le schéma universel de la base
  - Par raffinement successifs on obtient des sous relations sans perte d'informations et qui ne seront pas affectées lors des mises à jour (**non redondance**)



## Les formes normales

Dans une base de données relationnelle, une **forme normale** désigne un type de relation particulier entre les entités.

Le but essentiel de la normalisation est d'éviter les anomalies transactionnelles pouvant découler d'une mauvaise modélisation des données et ainsi éviter un certain nombre de problèmes potentiels tels que les anomalies de lecture, les anomalies d'écriture, la redondance des données et la contre-performance.

- **Notion intuitive de FN**

une « *bonne relation* » peut être considérée comme une **fonction** de la clé primaire vers les attributs restants

## 1ère Forme Normale 1FN

- Une relation est en 1FN si tout attribut est atomique (non décomposable)
- Contiennent des valeurs non répétitives (pas de liste, tableau .....

### Contre-exemple

ELEVE (**no\_elv**, nom, prenom, liste\_notes)

Un attribut ne peut pas être un ensemble de valeurs

### Décomposition

ELEVE (**no\_elv**, nom, prenom)

NOTE (no\_elv, no\_matiere, note)

- Constant dans le temps (age, Date de naissance)

## 2ème Forme Normale 2FN

Une relation est en 2FN si

- elle est en 1FN
- si tout attribut n'appartenant pas à la clé ne dépend pas d'une partie de la clé primaire mais de la totalité
  - C'est la phase d'identification des clés
  - Cette étape évite certaines redondances
  - Tout attribut doit dépendre fonctionnellement de la totalité de la clé

### Contre-exemple

une relation en 1FN qui n'est pas en 2FN

COMMANDE (date, no\_cli, no\_pro, qte, prixUHT)

elle n'est pas en 2FN car la clé = (date, no\_cli, no\_pro), et le prixUHT ne dépend que de no\_pro

### Décomposition

COMMANDE (date, no\_cli, no\_pro, qte)

PRODUIT (no\_pro, prixUHT)

## 3ème Forme Normale 3FN

Une relation est en 3FN si

- elle est en 2FN
- si tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non clé

Ceci correspond à la non transitivité des D.F. ce qui évite les redondances.  
En 3FN une relation préserve les D.F. et est sans perte.

### Contre-exemple

une relation en 2FN qui n'est pas en 3FN

VOITURE (**matricule**, marque, modèle, puissance)

on vérifie qu'elle est en 2FN ; elle n'est pas en 3FN car la clé = **matricule**, et la **puissance** dépend de (marque, modèle)

### Décomposition

VOITURE (**matricule**, marque, modèle)

MODELE (**marque, modèle**, puissance)

# **L'algèbre relationnelle**

## **I. Les opérations**

## **II. Le langage algébrique**

## I Les opérations

L'Algèbre relationnelle est une collection d'opérations

### OPÉRATIONS

- opérandes : 1 ou 2 relations
- résultat : une relation

### DEUX TYPES D'OPÉRATIONS:

#### → OPÉRATIONS ENSEMBLISTES

UNION

INTERSECTION

DIFFÉRENCE

#### → OPÉRATIONS SPÉCIFIQUES

PROJECTION

RESTRICTION

JOINTURE

DIVISION

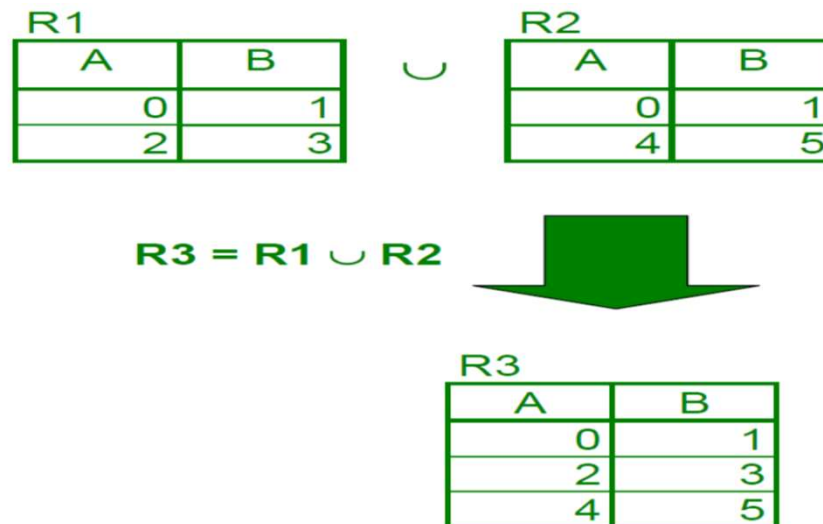


# UNION

L'union de deux relations R1 et R2 de même schéma est une relation R3 de schéma identique qui a pour n-uplets les n-uplets de R1 et/ou R2

On notera :

**$R3 = R1 \cup R2$  ou **UNION (R1,R2)****

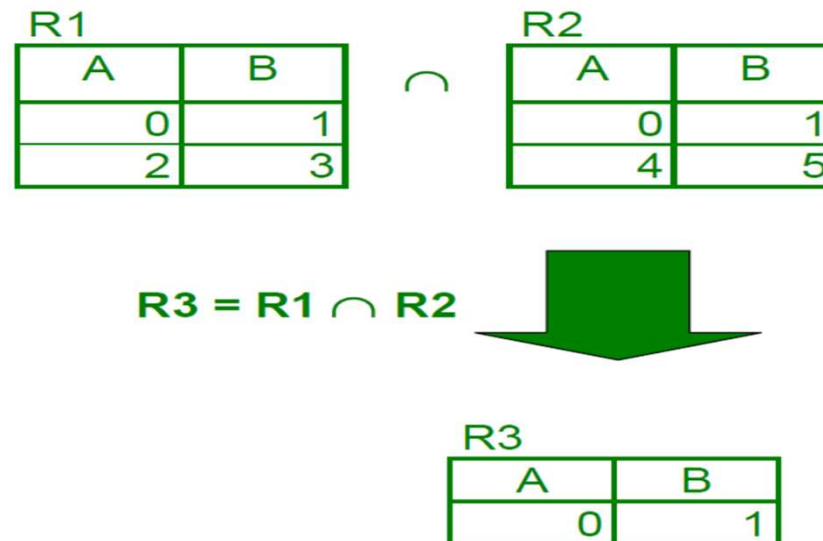


## INTERSECTION

L'intersection entre deux relations R1 et R2 de même schéma est une relation R3 de schéma identique ayant pour n-uplets les n-uplets communs à R1 et R2

On notera :

**$R3 = R1 \cap R2$  ou  $\text{Intersect}(R1, R2)$**



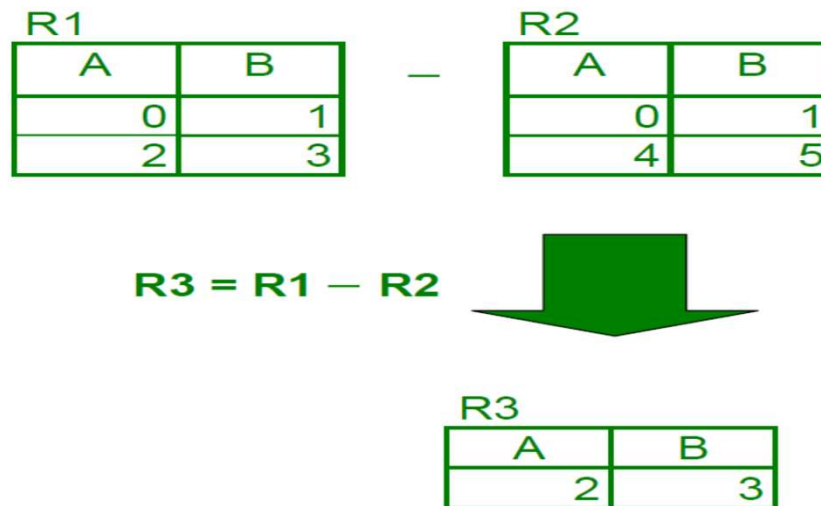


# DIFFÉRENCE

La différence entre deux relations R1 et R2 de même schéma est une relation R3 de schéma identique ayant pour n-uplets les n-uplets de R1 n'appartenant pas à R2

On notera :

$$R3 = R1 - R2 \text{ ou } \text{MINUS}(R1, R2)$$



# PROJECTION

La projection d'une relation R1 est la relation R2 obtenue en supprimant les attributs de R1 non mentionnés puis en éliminant éventuellement les nuplets identiques

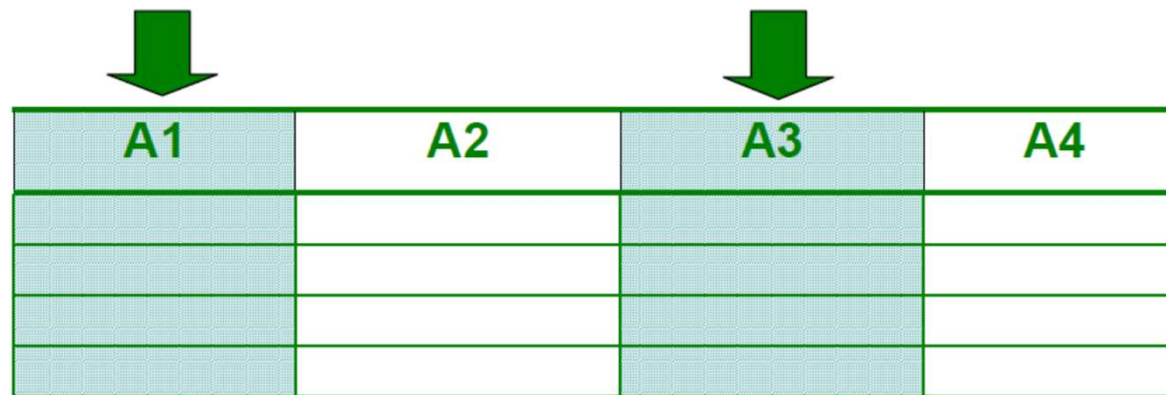
On notera :

$$R2 = \pi R1 (A_i, A_j, \dots, A_m)$$

la projection d'une relation R1 sur les attributs  $A_i, A_j, \dots, A_m$

→ La projection permet d'éliminer des attributs d'une relation

- Elle correspond à un découpage vertical :





A1	A2	A3	A4

## Requête 1 :

« Quels sont les références et les prix des produits ? »

**PRODUIT (IdPro, Nom, Marque, Prix)**



IdPro	Nom	Marque	Prix
P	PS1	IBM	1000
Q	Mac	Apple	2000
R	PS2	IBM	3000
S	Word	Microsoft	4000

$\pi$ PRODUIT (IdPro, Prix)



IdPro	Prix
P	1000
Q	2000
R	3000
S	4000

## Requête 2 :

« Quelles sont les marques des produits ? »

**PRODUIT (IdPro, Nom, Marque, Prix)**



IdPro	Nom	Marque	Prix
P	PS1	IBM	1000
Q	Mac	Apple	2000
R	PS2	IBM	3000
S	Word	Microsoft	4000

$\pi$ **PRODUIT (Marque)**



Marque
IBM
Apple
Microsoft

Notez l'élimination des doublons..

# RESTRICTION

La restriction d'une relation R1 est une relation R2 de même schéma n'ayant que les n-uplets de R1 répondant à la condition énoncée

On notera :

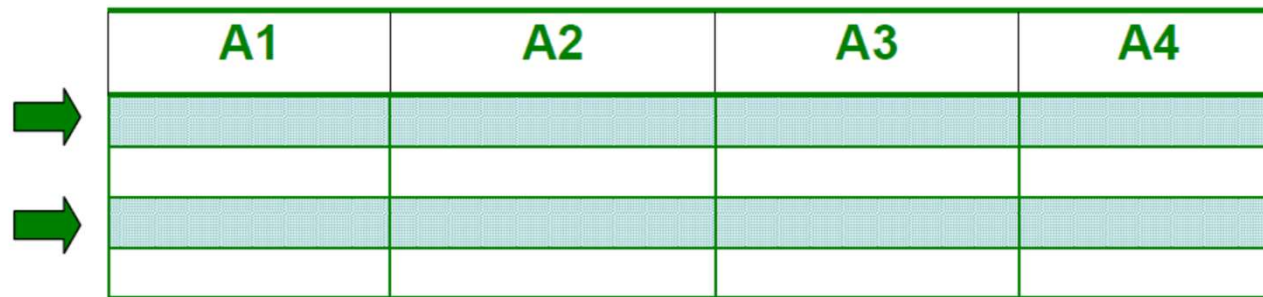
$$R2 = \sigma R1 (\text{condition})$$

la restriction d'une relation R1 suivant le critère "condition"

où "condition" est une relation d'égalité ou d'inégalité entre 2 attributs ou entre un attribut et une valeur

→ La restriction permet d'extraire les n-uplets qui satisfont une condition

- Elle correspond à un découpage horizontal :




A1	A2	A3	A4

### Requête 3 :

« Quelles sont les produits de marque 'IBM' ? »

**PRODUIT (IdPro, Nom, Marque, Prix)**



IdPro	Nom	Marque	Prix
P	PS1	IBM	1000
Q	Mac	Apple	2000
R	PS2	IBM	3000
S	Word	Microsoft	4000

$\sigma_{\text{PRODUIT (Marque = 'IBM')}}$



IdPro	Nom	Marque	Prix
P	PS1	IBM	1000
R	PS2	IBM	3000

# JOINTURE

La jointure de deux relations R1 et R2 est une relation R3 dont les n-uplets sont obtenus en concaténant les nuplets de R1 avec ceux de R2 et en ne gardant que ceux qui vérifient la condition de liaison

On notera :  **$R3 = R1 \times R2 \text{ (condition)}$**  ou  **$\text{JOIN}(R1, R2)(\text{condition})$**

la jointure de R1 avec R2 suivant le critère condition

- Le schéma de la relation résultat de la jointure est la concaténation des schémas des opérandes (s'il y a des attributs de même nom, il faut les renommer)
- Les n-uplets de  $R1 \times R2 \text{ (condition)}$  sont tous les couples (u1,u2) d'un n-uplet de R1 avec un n-uplet de R2 qui satisfont "condition"
- La jointure de deux relations R1 et R2 est le produit cartésien des deux relations suivi d'une restriction
- La condition de liaison doit être du type :

$\langle \text{attribut1} \rangle :: \langle \text{attribut2} \rangle$

où : attribut1  $\in$  1ère relation et attribut2  $\in$  2ème relation

:: est un opérateur de comparaison (égalité ou inégalité)

→ La jointure permet de composer 2 relations à l'aide d'un critère de liaison

**R1(A, B, C)**

A	B	C
A1	B1	10
A2	B2	10
A3	B3	20
A4	B4	30



**R2(U, V)**

U	V
10	V1
20	V2
30	V3

**R1 × R2 (R1.C = R2.U)**



A	B	C	U	V
A1	B1	10	10	V1
A1	B2	10	10	V1
A3	B3	20	20	V2
A4	B4	30	30	V3



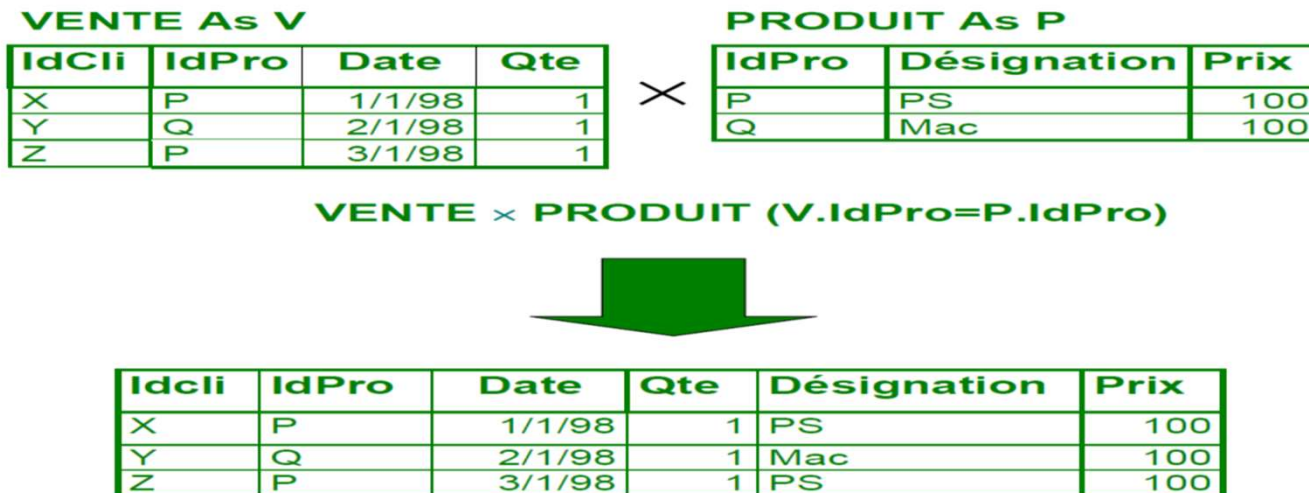
## Jointure naturelle

Jointure où l'opérateur de comparaison est l'égalité dans le résultat on fusionne les 2 colonnes dont les valeurs sont égales

→ La jointure permet d'enrichir une relation

### Requête 5 :

« Donnez pour chaque vente la référence du produit, sa désignation, son prix, le numéro de client, la date et la quantité vendue »



- La normalisation conduit à décomposer ; la jointure permet de recomposer

## Auto-jointure

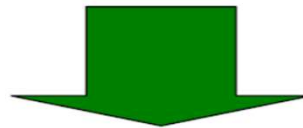
jointure d'une relation par elle-même

### Requête 6 :

« Quels sont les noms des clients qui habitent la même ville que John ? »

CLIENT As C1				CLIENT As C2		
IdCli	Nom	Ville		IdCli	Nom	Ville
X	Smith	Nice	×	X	Smith	Nice
Y	Blake	Paris		Y	Blake	Paris
Z	John	Nice		Z	John	Nice

**R1 = CLIENT × CLIENT (C1.Ville = C2.Ville)**



R1				
C1.IdCli	C1.Nom	Ville	C2.IdCli	C2.Nom
X	Smith	Nice	X	Smith
X	Smith	Nice	Z	John
Y	Blake	Paris	Y	Blake
Z	John	Nice	X	Smith
Z	John	Nice	Z	John

**R1**

C1.IdCli	C1.Nom	Ville	C2.IdCli	C2.Nom
X	Smith	Nice	X	Smith
X	Smith	Nice	Z	John
Y	Blake	Paris	Y	Blake
Z	John	Nice	X	Smith
Z	John	Nice	Z	John

$R2 = \sigma_{R1} (C2.Nom = 'John')$

**R2**

C1.IdCli	C1.Nom	Ville	C2.IdCli	C2.Nom
X	Smith	Nice	Z	John
Z	John	Nice	Z	John

$R3 = \pi_{R2} (C1.Nom)$

**R3**

C1.Nom
Smith
John

# DIVISION

Soit deux relations

$R1 (A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$

$R2 (B_1, B_2, \dots, B_m)$

Si le schéma de  $R2$  est un sous-schéma de  $R1$ .

La division de  $R1$  par  $R2$  est une relation  $R3$  dont :

- le schéma est le sous-schéma complémentaire de  $R2$  par rapport à  $R1$
- un  $n$ -uplet  $(a_1, a_2, \dots, a_n)$  appartient à  $R3$  si  $(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m)$  appartient à  $R1$  pour tous  $(b_1, b_2, \dots, b_m) \in R2$ .

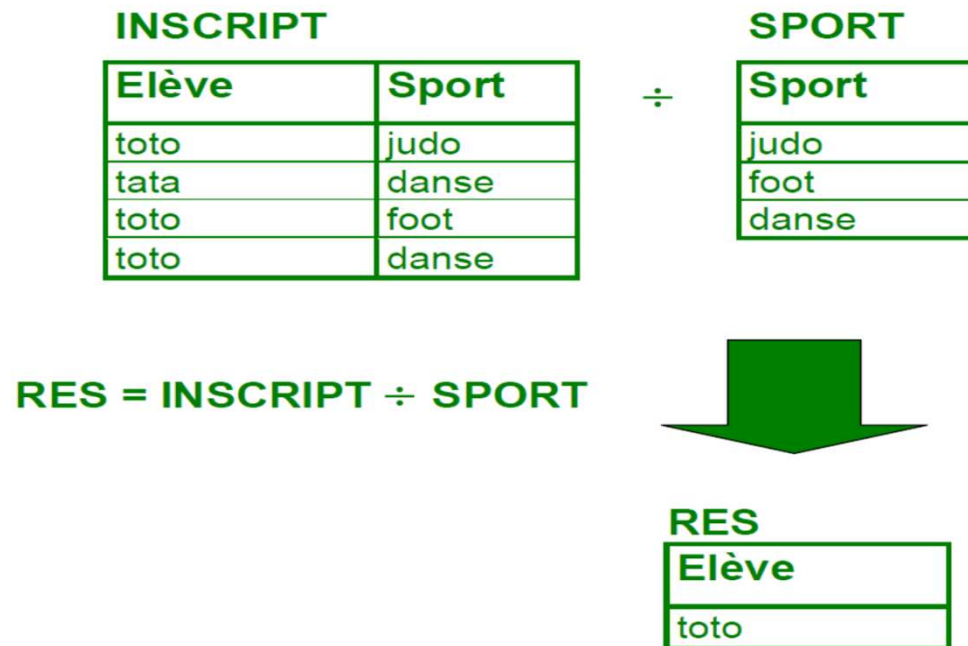
On notera :  **$R3 = R1 \div R2$**

la division de  $R1$  par  $R2$

la division permet de rechercher dans une relation les sous n-uplets qui sont complétés par tous ceux d'une autre relation Elle permet de répondre à des questions qui sont formulées avec le quantificateur universel : « *pour tout ...* »

### Requête 6 :

« Quels sont les élèves qui sont inscrits à tous les sports ? »



Requêtes sur le schéma CLIENT, PRODUIT, VENTE

CLIENT (**IdCli**, nom, ville)

PRODUIT (**IdPro**, désignation, marque, prix)

VENTE (**IdCli**, **IdPro**, **date**, qte)

Requête 1 :

Donner les no des produits de marque Apple et de prix < 5000 Dhs

Requête 2 :

Donner les no des clients ayant acheté un produit de marque Apple

Requête 3 :

Donner les no des clients n'ayant acheté que des produits de marque Apple

Requête 4

Donner les no des clients ayant acheté tous les produits de marque Apple

## Le langage algébrique

Le langage algébrique permet de formuler une question par une suite d'opérations de l'algèbre relationnelle

Requêtes sur le schéma CLIENT, PRODUIT, VENTE

CLIENT (**IdCli**, nom, ville)

PRODUIT (**IdPro**, désignation, marque, prix)

VENTE (**IdCli**, **IdPro**, **date**, qte)

**Requête 8 :**

« Donner les no des produits de marque Apple et de prix < 5000 Dhs »

**$R1 = \sigma_{\text{PRODUIT}} (\text{marque} = \text{'Apple'})$**

**$R2 = \sigma_{\text{PRODUIT}} (\text{prix} < 5000)$**

**$R3 = R1 \cap R2$**

**$\text{RESUL} = \pi_{\text{R3}} (\text{IdPro})$**

**Requête 9 :**

« Donner les no des clients ayant acheté un produit de marque Apple »

**$R1 = \sigma_{\text{PRODUIT (marque = 'Apple')}}}$**

**$R2 = R1 \times \text{VENTE (} R1.\text{IdPro} = \text{VENTE.IdPro)}$**

**$\text{RESUL} = \pi R2 (\text{IdCli})$**



### **Requête 10 :**

« Donner les no des clients n'ayant acheté que des produits de marque Apple »

**$R1 = \text{VENTE} \times \text{PRODUIT} (\text{VENTE.IdPro} = \text{PRODUIT.IdPro})$**

**$R2 = \sigma R1 (\text{marque} = \text{'Apple'})$**

**$R3 = \pi R2 (\text{IdCli})$**

**$R4 = \sigma R1 (\text{marque} \neq \text{'Apple'})$**

**$R5 = \pi R4 (\text{IdCli})$**

**$\text{RESUL} = R3 - R5$**

**Requête 11 :**

« Donner les no des clients ayant acheté tous les produits de marque Apple »

**$R1 = \sigma_{\text{PRODUIT (marque = 'Apple')}}$**

**$R2 = \pi R1 (\text{IdPro})$**

**$R3 = \pi \text{VENTE (IdCli, IdPro)}$**

**$R4 = R3 \div R2$**

# Les structures physiques

Les tables sont stockées sur le disque de l'ordinateur.

Si elles sont très volumineuses, l'accès aux données et leur modification risquent de prendre un temps considérable.

*Exemple :*

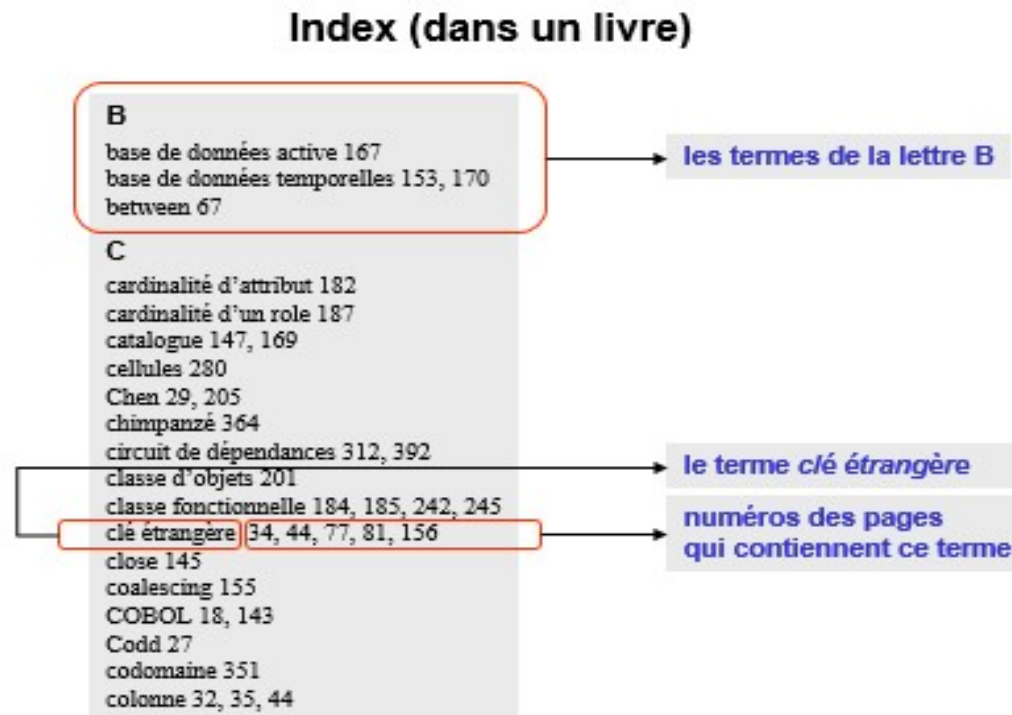
la lecture d'une table de 2.500.000 de lignes de 400 octets prend près d'**une minute** dans le meilleur des cas et **une heure** dans le cas contraire.

Les **structures physiques** garantissent de bonnes performances aux opérations de lecture et de modification.

Deux mécanismes principaux :

- les index
- les espaces de stockage

# Les structures physiques - Les index



# Les structures physiques - Les index

## Index (dans une base de données)



# Les structures physiques - Les index

## Index

L'accès à une ligne d'une table via un index prend généralement de **10 à 20 millisecondes**.

En l'absence d'index, l'accès à cette ligne peut exiger la lecture de toute la table, soit de **1 minute à 1 heure** !

## Les structures physiques - Les index

1. Quel est l'intérêt d'un index dans une base de données?
2. Qu'est ce qu'un index de base de données?
3. Comment créer un index?
4. Quels sont les types d'index?

## Quel est l'intérêt d'un index dans une base de données?

1. Le moteur de SGBD utilise les index pour rechercher rapidement les données
2. Si les index n'existent pas, le moteur SGBD parcourt tous les enregistrements de la table
3. L'impact des index sur les requêtes d'écriture est moins important



# Qu'est ce qu'un index de base de données?

1. Une base de données sans index oblige le moteur du SGBD de parcourir de A à Z pour fournir le résultat
2. Les SGBDR créent automatiquement un index sur la clé primaire
3. Les SGBDR créent automatiquement un index la contrainte d'unicité (UNIQUE) dans une table
4. Il n'y a pas d'index crée automatiquement par le SGBD derrière une clé étrangère(FOREIGN KEY)

# Comment créer un index?

La syntaxe générale de l'ordre SQL de création d'un index:

**CREATE INDEX**<nom index>

**ON** <nom\_table>(<liste\_colones>)

## **Exemple 1: création d'un index sur une seule colonne:**

```
CREATE INDEX idx_nomProduit
```

```
ON produit (NomProduit)
```

## **Exemple 2: création d'un index sur deux colonnes:**

```
CREATE INDEX idx_nomCategorie
```

```
ON produit (NomProduit, categorie)
```

## **Exemple 3: création d'un index unique:**

```
CREATE UNIQUE INDEX idx_refProduit
```

```
ON produit (RefProduit)
```

# Quels sont les types d'index?

1. B-Tree(arbre équilibré)
  2. Hash (index de hachage)
- Chaque type d'index utilise un algorithme qui convient à un type particulier de requêtes.

# Les structures physiques

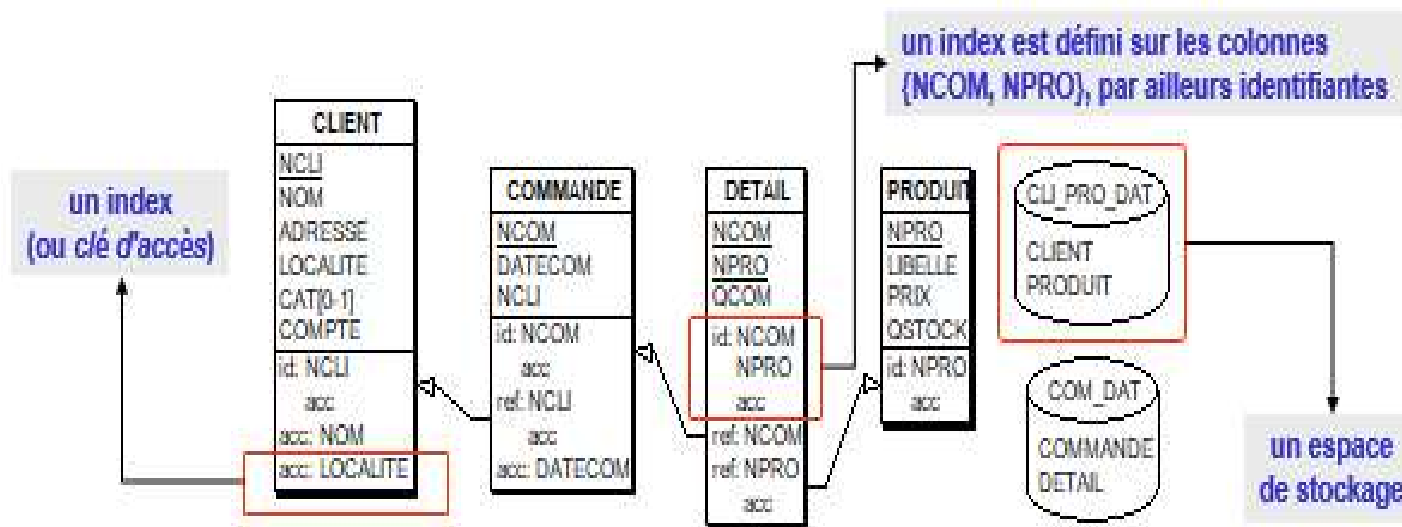
## Espace de stockage

La table est une collection de lignes dont les éléments doivent être stockés sur un disque. Les lignes seront rangées dans un espace spécial qui leur est réservé : un **espace de stockage**. L'espace correspond à un *fichier* occupant tout ou partie d'un disque (voire de plusieurs disques).

Un **espace de stockage** est caractérisé notamment par son adresse, son volume initial, la manière dont il grandit ou se réduit selon les besoins, les tables dont il accueille les lignes, la technique de rangement des lignes.

# Les structures physiques

## Un schéma physique



# **Le langage SQL**

# Définition

SQL (Structured Query Language) est le langage de programmation utilisé pour définir et manipuler des bases de données relationnelles organisées sous forme de tables contenant des lignes et des colonnes.

# Définition

SQL est à la fois un **langage de définition de données** et un **langage de manipulation de données** :

- D'une part, en tant que langage de définition de données, SQL permet d'implémenter physiquement un modèle relationnel au sein d'un système de gestion de bases de données en offrant des instructions qui permettent de créer des tables et des colonnes.
- D'autre part, en tant que langage de manipulation de données, SQL offre une multitude d'opérations permettant de faire la recherche, l'insertion, la suppression et la mise à jour de données.



## Langage de définition de données

- **Création d'une table**

L'instruction SQL qui permet de créer une table s'appelle CREATE TABLE. Sa syntaxe est la suivante :

```
CREATE TABLE [Nom_De_La_table] (  
    /* Definition des colonnes de la table */  
    [Colonne] [Type] [Contraintes],  
    [Colonne] [Type] [Contraintes],  
    ...  
    [Colonne] [Type] [Contraintes],  
    /* Contraintes multicolonnes (si besoin) */  
    [Contraintes_multicolonnes]  
);
```

L'instruction **CREATE TABLE** permet de créer une table et de définir le nom, le type de données et les éventuelles contraintes d'intégrité de chaque colonne.

# Types de données

L'ensemble des types de données utilisés dans les différents SGBDs du marché varie légèrement d'un SGBD à un autre.

## type numériques

type	taille	Range (Signé)	Range (non signé)	utilisation
TINYINT	1 octet	(-128.127)	(0255)	petites valeurs entières
SMALLINT	2 octets	(768,32 -32 767)	(535 0,65)	valeur entière
MEDIUMINT	3 octets	(-8388 608,8 388 607)	(0,16 777215)	valeur entière
INT ou INTEGER	4 octets	(-2 147 483 648,2 147 483 647)	(0,4 294 967 295)	valeur entière
BIGINT	8 octets	(-9.233.372.036.854.775 808,9 223.372.036.854.775 807)	(0,18 446.744.073.709.551 615)	Valeur maximale entier
FLOAT	4 octets	(-3,402 823 466 E + 38,1.175 494 351 E -38), 0, (1.175 494 351 E-38,3.402 823 466 351 E + 38)	0, (1.175 494 351 E-38,3.402 823 466 E + 38)	Simple précision valeurs à virgule flottante
DOUBLE	8 octets	(1.797 693 134 862 315 7 E + 308,2.225 073 858 507 201 4 E-308), 0, (2.225 073 858 507 201 4 E-308,1.797 693 134 862 315 7 E + 308)	0, (2.225 073 858 507 201 4 E-308,1.797 693 134 862 315 7 E + 308)	Double-precision valeurs à virgule flottante
DECIMAL	De DECIMAL (M, D), si M> D, M + 2 est par ailleurs D + 2	Cela dépend des valeurs de M et D	Cela dépend des valeurs de M et D	valeur décimale

# Types de données

## Date et heure Types

type	taille (Byte)	portée	format	utilisation
DATE	3	01.01.1000 / 9999-12-31	AAAA-MM-JJ	Les valeurs de date
TIME	3	'-838: 59: 59' / '838: 59: 59'	HH: MM: SS	Valeur temps ou la durée
ANNÉE	1	1901/2155	AAAA	Année Valeur
DATETIME	8	1000-01-0100: 00: 00 / 9999-12-31 23:59:59	AAAA-MM-JJ HH: MM: SS	Mixage valeurs date et heure
TIMESTAMP	4	Parfois 00/2037 Année: 1970-01-01 00:00	AAAAMMJJ HHMM SS	Date de mélange et la valeur temps, un horodatage

# Types de données

## type String

type	taille	utilisation
CHAR	0-255 octets	chaîne longueur fixe
VARCHAR	0-65535 octets	chaînes de longueur variable
TINYBLOB	0-255 octets	Pas plus de 255 caractères dans une chaîne binaire
TINYTEXT	0-255 octets	Courtes chaînes de texte
BLOB	0-65535 octets	données textuelles longues sous forme binaire
TEXTE	0-65535 octets	Longue données de texte
MEDIUMBLOB	0-16777215 octets	forme binaire de longueur moyenne des données de texte
MEDIUMTEXT	0-16777215 octets	longueur moyenne des données de texte
LONGBLOB	0-4294967295 octets	Grands données de texte sous forme binaire
LONGTEXT	0-4294967295 octets	Grande données de texte

## Contraintes d'intégrité

- Une contrainte d'intégrité est un mécanisme qui s'assure que les valeurs d'une colonne donnée soient toujours cohérentes.
- quatre contraintes d'intégrités seront prises en considération :
  1. PRIMARY KEY : Déclare la colonne comme étant la clé primaire de la table. Cette contrainte peut être utilisée dans sa version multi-colonne pour définir une clé primaire composée de plusieurs attributs.
  2. NOT NULL : S'assure que la colonne ne contienne pas de valeurs NULL.
  3. CHECK(C) : S'assure que toutes les valeurs de la colonne satisfont la condition C.
  4. REFERENCES Tab(Col) : Déclare la colonne comme clé étrangère qui référence la colonne Col de la table Tab.

## Exemple de création de tables

- Les instructions respectives de création des tables country, city et countryLanguage de la base de données World sont données dans ce qui suit :

```
CREATE TABLE country (  
  Code TEXT PRIMARY KEY,  
  Name TEXT NOT NULL,  
  Continent TEXT NOT NULL,  
  SurfaceArea REAL NOT NULL CHECK (SurfaceArea > 0),  
  Population INTEGER NOT NULL CHECK (Population > 0),  
  HeadOfState TEXT NOT NULL,  
  Capital INTEGER NOT NULL REFERENCES city (ID)  
);
```

La table country contient sept colonnes :

— **Code** : est l'identifiant unique d'un pays e.g. 'FIN' pour Finlande, etc. Son type est TEXT et c'est la clé primaire de la table country.

— **Name** : est le nom du pays e.g. 'Finland', 'Sweden'. Son type est TEXT et sa valeur ne peut pas être NULL.

— **Continent** : est le nom du continent où se trouve le pays e.g. 'Africa', 'Europe', etc. Son type est TEXT et sa valeur ne peut pas être NULL.

— **SurfaceArea** : est la superficie du pays. Son type est REAL et sa valeur est strictement positive et ne peut pas être NULL.

— **Population** : est la population du pays. Son type est INTEGER et sa valeur est strictement positive et ne peut pas être NULL.

— **HeadOfState** : est le nom du chef d'état du pays e.g. 'Vladimir Putin' pour la Russie. Son type est TEXT et sa valeur ne peut pas être NULL.

— **Capital** : est la capitale du pays. Son type est INTEGER et c'est une clé étrangère qui référence la colonne ID de la table city. Sa valeur ne peut pas être NULL.

## Exemple de création de tables

```
CREATE TABLE city (  
  ID INTEGER PRIMARY KEY,  
  Name TEXT NOT NULL,  
  CountryCode TEXT NOT NULL REFERENCES country (Code),  
  Population INTEGER NOT NULL CHECK (Population > 0)  
);
```

La table city contient quatre colonnes :

— **ID** : est l'identifiant unique d'une ville. Son type est INTEGER et c'est la clé primaire de la table city.

— **Name** : est le nom de la ville e.g. 'New York City', etc. Son type est TEXT et sa valeur ne peut pas être NULL.

— **countryCode** : est le code du pays où se situe la ville. Son type est TEXT, c'est une clé étrangère qui référence la colonne Code de la table country. Sa valeur ne peut pas être NULL.

**Population** : est la population de la ville. Son type est INTEGER, sa valeur est strictement positive et ne peut pas être NULL.

## Exemple de création de tables

```
CREATE TABLE countrylanguage (  
  CountryCode TEXT NOT NULL REFERENCES country(Code),  
  Language TEXT NOT NULL,  
  PRIMARY KEY (CountryCode, Language)  
);
```

La table countryLanguage contient deux colonnes :

— **CountryCode** : est le code du pays où la langue est parlée. Son type est TEXT, c'est une clé étrangère qui référence la colonne Code de la table country et elle ne peut pas être NULL.

— **Language** : est le nom de la langue parlée dans le pays dont le code est countryCode e.g. 'Arabic', 'Berberi' pour le Maroc. Son type est TEXT et sa valeur ne peut pas être NULL.

- La clé primaire de la table countryLanguage est composée des deux colonnes CountryCode et Language.



## Suppression d'une table

- L'instruction SQL qui permet de supprimer une table s'appelle DROP TABLE. Sa syntaxe est la suivante :
- ***DROP TABLE*** [ Nom\_de\_la\_table ];

### Exemple de suppression d'une table

Les instructions suivantes permettent de supprimer les tables country, city et countryLanguage:

```
DROP TABLE country ;
```

```
DROP TABLE city ;
```

```
DROP TABLE countrylanguage ;
```

# Langage de manipulation de données

- **Insertion de données**

L'instruction SQL qui permet d'insérer des données dans une table s'appelle INSERT INTO. Sa syntaxe est la suivante :

```
INSERT INTO [ Nom_de_la_table ]
```

```
VALUES (v1 ,v2 , ... );
```

Où v1,v2, ... sont les valeurs du tuple à ajouter dans la table. Ces valeurs doivent être données dans l'ordre dans lequel les colonnes ont été déclarées lors de l'utilisation de l'instruction CREATE TABLE

## Exemples d'insertion de données

Afin d'illustrer l'utilisation de l'instruction INSERT INTO, ce qui suit est un exemple d'ajout de la ville de Tanger dans la table city :

```
INSERT INTO city
```

```
VALUES (35 , 'Tanger ', 'MA ', 2168000);
```

Ce qui suit est un autre exemple d'ajout dans la table countryLanguage des langues parlées au Maroc:

```
INSERT INTO countrylanguage
```

```
VALUES ('MA ', 'Arabic ');
```

```
INSERT INTO countrylanguage
```

```
VALUES ('MA ', 'Berberi ');
```

## Suppression de données

L'instruction SQL qui permet de supprimer des données dans une table s'appelle DELETE FROM. Sa syntaxe est la suivante :

```
DELETE FROM [ Nom_de_la_table ]
```

```
WHERE [ Condition ];
```

Cette instruction supprime tous les tuples d'une table qui vérifient la condition qui suit le mot-clé WHERE.

## Exemple de suppression de données

Afin d'illustrer l'utilisation de l'instruction DELETE FROM, ce qui suit est un exemple de suppression de tous les pays d'Afrique :

```
DELETE FROM country
```

```
WHERE Continent = 'Africa ';
```

## Mise à jour de données

L'instruction SQL qui permet de mettre à jour les données d'une table s'appelle UPDATE. Sa syntaxe est la suivante :

**UPDATE** [ Nom\_de\_la\_table ]

**SET** [ Colonne ] = [ Valeur ]

**WHERE** [ Condition ];

L'instruction UPDATE effectue la modification qui suit le mot clé SET sur tous les tuples de la table qui vérifient la condition qui suit le mot clé WHERE.

## Exemple de mise à jour de données

Afin d'illustrer l'utilisation de l'instruction UPDATE, ce qui suit est un exemple qui fait la mise à jour suivante « Donald Trump est le nouveau président des USA » :

```
UPDATE country
```

```
SET HeadOfState = 'Donald _ Trump '
```

```
WHERE Code = 'USA ';
```

## Recherche de données

La recherche de données en SQL se fait de manière similaire à celle de l'algèbre relationnel dans la mesure où elle s'appuie entre autres sur les opérations de projection, de restriction et de jointure. L'instruction SQL qui permet d'effectuer une recherche de données s'appelle SELECT. Sa syntaxe est la suivante :

**SELECT** [ Colonne ], [ Colonne ], ...

**FROM** [ Table ], [ Table ], ...

**WHERE** [ Condition ];

— la **projection** sur les colonnes qui suivent le mot-clé SELECT (Le symbole \* peut être utilisée pour projeter sur toutes les colonnes).

— du **produit cartésien** des tables qui suivent le mot-clé FROM

— et **restreint** le résultat aux tuples qui satisfont la condition qui suit le mot-clé WHERE. Les opérateurs logique AND et OR peuvent être utilisés pour construire la condition



## Exemples de requêtes d'interrogations d'une seule table

Afin d'illustrer l'utilisation de l'instruction SELECT, nous allons traduire des requêtes exprimées en langage naturel vers des requêtes exprimées en SQL :

**Donner le nom de chaque pays et le nom de son chef d'état.**

```
SELECT Name , HeadOfState
```

```
FROM country ;
```

**Donner le nom de chaque pays et le code de sa capitale.**

```
SELECT Name , Capital
```

```
FROM country ;
```

**Donner le nom des pays dont la superficie dépasse 1000000m<sup>2</sup>.**

```
SELECT Name
```

```
FROM country
```

```
WHERE SurfaceArea > 1000000;
```

## Exemples de requêtes d'interrogations d'une seule table

— Donner le nom de tous les pays d'Europe.

```
SELECT Name
FROM country
WHERE Continent = 'Europe';
```

— Donner le nom des pays africains dont la superficie est inférieur à 500000m<sup>2</sup>.

```
SELECT Name
FROM country
WHERE Continent = 'Africa'
AND SurfaceArea < 500000;
```

— Qui est le chef d'état du Zimbabwe ?

```
SELECT HeadOfState
FROM country
WHERE Name = 'Zimbabwe';
```

— Quelle est la superficie de l'Algérie ?

```
SELECT SurfaceArea
FROM country
WHERE Name = 'Algeria';
```

— Donner les langues parlées en Suisse sachant que ce pays porte le code CHE.

```
SELECT Language
FROM countryLanguage
WHERE countryCode = 'CHE';
```

## Exemples de requêtes d'interrogations sur plusieurs tables

**Donner le nom de toutes les villes d'Algérie.** Cette requête nécessite de faire une jointure entre les tables city et country. Elle peut être exprimée de la manière suivante :

```
SELECT city . Name
```

```
FROM country , city
```

```
WHERE CountryCode = Code
```

```
AND country . Name = 'Algeria ';
```

## Les remarques concernant la recherche de données à travers plusieurs tables

### Remarque1

La jointure est exprimée comme une condition du WHERE. C'est-à-dire qu'il n'est pas nécessaire de faire appel à une autre instruction SQL pour faire la jointure. En outre, les éventuelles restrictions peuvent être exprimées avant (ou après) la jointure en utilisant l'opérateur AND. Cette manière d'exprimer la jointure comme une restriction découle du fait qu'en algèbre relationnelle, une jointure peut être vue comme un produit cartésien suivi d'une restriction

## Les remarques concernant la recherche de données à travers plusieurs tables

### Remarque2

Il est éventuellement nécessaire de désambiguïser le nom des colonnes. En effet, dans l'exemple précédent, la colonne Name existe dans deux tables différentes : city et country. Pour faire la différence entre ces deux colonnes, il est nécessaire de précéder le nom de la colonne par le nom de la table suivi d'un point. Par exemple city.Name pour la colonne Name de la table city et country.Name pour la colonne Name de la table country.

## Exemples de requêtes d'interrogations sur plusieurs tables

- Donner le nom de toutes les villes européennes ainsi que le pays dans lequel se trouve chaque ville.

```
SELECT city.Name, country.Name
FROM city, country
WHERE countryCode = Code
AND Continent = 'Europe';
```

- Donner le nom de toutes les capitales européennes ainsi que le pays dans lequel se trouve chaque capitale.

```
SELECT city.Name, country.Name
FROM city, country
WHERE ID = Capital
AND Continent = 'Europe';
```

- Donner le nom de tous les pays africains francophones.

```
SELECT Name
FROM country, countryLanguage
WHERE Code = CountryCode
AND Continent = 'Africa'
AND Language = 'French';
```

- Quelles sont les langues parlées en Suisse (Switzerland) ?

```
SELECT Language
FROM country, countryLanguage
WHERE Code = CountryCode
AND Name = 'Switzerland';
```

- Quelles sont les langues parlées dans le pays dont la capitale est Londres (London) ?

```
SELECT Language
FROM city, countryLanguage, country
WHERE countryLanguage.countryCode = Code
AND ID = Capital
AND city.Name = 'London';
```

## Opérations ensemblistes

À l'instar de l'algèbre relationnelle, le langage SQL offre les trois principales opérations ensemblistes vues dans le chapitre qui porte sur l'algèbre relationnelle, à savoir :

- L'union en utilisant le mot-clé UNION ;
- L'intersection en utilisant le mot-clé INTERSECT ;
- La différence en utilisant le mot-clé EXCEPT.

Ces trois opérations peuvent être utilisées entre deux instructions SELECT. Et comme en algèbre relationnelle, les deux instructions SELECT doivent avoir les mêmes colonnes, c'est-à-dire, la même projection.

# Opérations ensemblistes

La syntaxe de l'utilisation des opérations ensemblistes est la suivante :

**SELECT** [ Colonne ], [ Colonne ], ...

**FROM** [ Table ], [ Table ], ...

**WHERE** [ Condition ];

## opérations ensemblistes

**SELECT** [ Colonne ], [ Colonne ], ...

**FROM** [ Table ], [ Table ], ...

**WHERE** [ Condition ];



## Exemples :

**Donner le nom des pays d'Afrique et d'Asie.** Nous pouvons diviser cette requête en deux requêtes : la première retourne les pays d'Afrique et la seconde retourne les pays d'Asie. Puis, nous devons faire l'union des résultats des deux requêtes :

```
SELECT Name  
FROM country  
WHERE Continent = 'Africa '  
UNION  
SELECT Name  
FROM country  
WHERE Continent = 'Asia ';
```

## Exemples :

- Quelles sont les villes du royaume uni (United Kingdom) qui ont des homonymes aux Canada?

```
SELECT city.Name
FROM city, country
WHERE CountryCode = Code
AND country.Name = 'United_Kingdom'
INTERSECT
SELECT city.Name
FROM city, country
WHERE CountryCode = Code
AND country.Name = 'Canada';
```

## Exemples :

- Quelles sont les langues parlées à la fois en Suisse (Switzerland) et en Belgique (Belgium)?

```
SELECT Language
FROM country, countryLanguage
WHERE country.Code = countryLanguage.CountryCode
AND Name = 'Switzerland'
INTERSECT
SELECT Language
FROM country, countryLanguage
WHERE country.Code = countryLanguage.CountryCode
AND Name = 'Belgium';
```

## Exemple

- Quelles sont les langues parlées uniquement en Afrique du sud (South Africa)?

```
SELECT Language
FROM countrylanguage, country
WHERE CountryCode = Code
AND Name = 'South_Africa'
EXCEPT
SELECT Language
FROM countrylanguage, country
WHERE CountryCode = Code
AND Name != 'South_Africa';
```

## Exemple

- Donner le nom des pays qui parlent uniquement l'espagnol (Spanish).

```
SELECT Name
FROM countrylanguage, country
WHERE CountryCode = Code
AND Language = 'Spanish'
EXCEPT
SELECT Name
FROM countrylanguage, country
WHERE CountryCode = Code
AND Language != 'Spanish';
```

## Limitation des résultats

Parfois, nous pouvons avoir besoin d'afficher seulement d'un échantillon de tuples (ou un certain nombre de tuples) et non pas la totalité des tuples d'une requête SELECT. Le langage SQL offre pour cela l'instruction LIMIT dont la syntaxe est la suivante :

**SELECT** [ Colonne ] , [ Colonne ] , ...

**FROM** [ Table ] , [ Table ] , ...

**WHERE** [ Condition ]

**LIMIT** [ Nombre\_maximum\_de\_tuples ] ;

L'instruction LIMIT limite le résultat de l'instruction SELECT à un nombre de tuples égal au maximum au nombre qui suit le mot-clé LIMIT.

## Exemple

- Donner le nom de 5 pays d'Afrique.

```
SELECT Name
```

```
FROM country
```

```
WHERE Continent = 'Africa '
```

```
LIMIT 5;
```

## Tri des résultats

Contrairement à l'algèbre relationnelle, SQL offre la possibilité de trier les résultats retournés par instruction SELECT. Le tri se fait selon l'ordre croissant ou décroissant des valeurs d'une colonne. L'instruction qui permet de trier les résultats s'appelle ORDER BY. Sa syntaxe est la suivante :

**SELECT** [ Colonne ], [ Colonne ], ...

**FROM** [ Table ], [ Table ], ...

**WHERE** [ Condition ]

**ORDER BY** [ Colonne ] **ASC** / **DESC** ;

L'instruction ORDER BY trie le résultat de l'instruction SELECT selon l'ordre croissant (ASC) ou décroissant (DESC) des valeurs de la colonne qui suit le mot-clé ORDER BY.



## Exemples

- Donner le nom et la population des pays du monde selon l'ordre décroissant de leur population.

```
SELECT Name, Population
FROM country
ORDER BY Population DESC;
```

- Donner le top 5 des pays les plus peuplés du monde.

```
SELECT Name
FROM country
ORDER BY Population DESC
LIMIT 5;
```

- Donner le top 3 des pays les plus grands d'Afrique.

```
SELECT Name
FROM country
WHERE Continent = 'Africa'
ORDER BY SurfaceArea DESC
LIMIT 3;
```

- Quel est le pays le plus petit au monde et quelle est sa superficie ?

```
SELECT Name, SurfaceArea
FROM country
ORDER BY SurfaceArea ASC
LIMIT 1;
```

## Exemple

- Donner les cinq capitales les plus peuplées du monde ainsi que le pays où elles se trouvent ?

```
SELECT city.Name, country.Name  
FROM country, city  
WHERE Id = Capital  
ORDER BY city.Population DESC  
LIMIT 5;
```

## Fonctions

- Le langage SQL offre plusieurs fonctions pour effectuer des traitements sur les données. Le tableau suivant donne la signification de chacune de ces fonctions.

Fonction	Signification
COUNT	Retourne le nombre de ligne d'une colonne donnée
MAX	Retourne la plus grande valeur d'une colonne donnée
MIN	Retourne la plus petite valeur d'une colonne donnée
AVG	Retourne la moyenne des valeurs d'une colonne donnée
SUM	Retourne la somme des valeurs d'une colonne donnée

## Fonctions

L'appel d'une de ces fonction doit se faire directement après le mot-clé SELECT et ne peut être utilisé que sur une et une seule colonne de projection. La syntaxe de l'utilisation des fonctions est la suivante :

**SELECT FONCTION** ([ Colonne ])

**FROM** [ Table ], [ Table ], ...

**WHERE** [ Condition ]

# Exemples

— Combien de pays y a-t-il en Afrique ?

```
SELECT COUNT(Code)
FROM country
WHERE Continent = 'Africa';
```

— Quelle est la superficie du plus grand pays au monde ?

```
SELECT MAX(SurfaceArea)
FROM country;
```

— Quelle est la superficie du plus petit pays d'Europe ?

```
SELECT MIN(SurfaceArea)
FROM country
WHERE Continent = 'Europe';
```

— Quelle est la population moyenne des pays d'Asie ?

```
SELECT AVG(Population)
FROM country
WHERE Continent = 'Asia';
```

— Quelle est la population d'Afrique ?

```
SELECT SUM(Population)
FROM country
WHERE Continent = 'Africa';
```

***FIN DU MODULE***

## Révision

### Select mono-table

- **NULL**
- NULL signifie : « **non renseigné** ».
- C'est la seule information codée qu'on rentre dans une table.
- La valeur « 0 », par contre, ne signifie pas du tout « non renseignée », mais bien « valeur = 0 », comme on dirait « valeur = 500 ».
- Quand un attribut **peut valoir NULL**, on dit qu'il n'est « **pas obligatoire** ».
- Quand un attribut **ne peut pas valoir NULL**, on dit qu'il est « **obligatoire** » ou « **NOT NULL** ».

# Classification des commandes du SQL et manuel de référence

- 4 types de commandes

Les commandes du SQL se divisent en 4 sous-ensembles :

- Le **DDL** : data definition language
- Le **DML** : data manipulation language
- Le **DSL** : data select language (formulation non standard)
- Le **DCL** : data control language



# Le DDL : commandes des tables : **CREATE, ALTER, DROP**

- Ce sont les commandes qui vont permettre de créer, modifier, détruire les tables : **CREATE TABLE, ALTER TABLE, DROP TABLE.**
- Ces commandes permettront aussi de créer, modifier et supprimer d'autres objets de la BD : les BD, les séquences (auto-incréments ORACLE), les index, les vues, etc. **CREATE DATABASE, CREATE VIEW, CREATE INDEX,** etc

# Le DML : commandes des tuples : INSERT, UPADTE, DELETE

- Ce sont les commandes qui vont permettre de créer, modifier, détruire les tuples.

# Le DSL : SELECT : l'algèbre relationnelle

- Le SELECT est la commande qui permet de consulter les données de la BD.
- Le SELECT met en oeuvre l'algèbre relationnelle.

# Le DCL : commandes de contrôle

- La commande **CREATE USER** permet de créer des utilisateurs qui pourront se connecter à la base de données. **DROP USER** et **ALTER USER** permettront la suppression et la modification.
- La commande **GRANT** permet de donner des droits aux utilisateurs : droits de création, modification, suppression de tables, et droits de création, modification, suppression et consultation de tuples. Cette commande permet aussi de créer des utilisateurs.
- La commande **REVOKE** permet de supprimer les droits créés par la commande GRANT.
- La commande **COMMIT** permet de valider les modifications dans la BD (les commandes du DML). Selon les environnements, les modifications peuvent être validées automatiquement par défaut ou pas (variable d'environnement AUTOCOMMIT à vrai ou à faux).
- La commande **ROLLBACK** permet au contraire de revenir en arrière, s'il n'y a pas eu de validation manuelle ou automatique

# SQL : CONSULTATION DE LA BASE DE DONNEES

## PRINCIPALES NOTIONS

- |                                   |                                 |
|-----------------------------------|---------------------------------|
| • <b>Projection - Restriction</b> | <b>SELECT / FROM / Where</b>    |
| • <b>Distinct</b>                 | <b>year / month / day</b>       |
| • <b>Tri</b>                      | <b>length / substr / concat</b> |
| • <b>Attribut calculé</b>         | <b>in, between, like</b>        |
| • <b>Projection primaire</b>      | <b>Order by / asc / desc</b>    |

# La commande de recherche : le select

La commande SELECT permet de faire des opérations de recherche et de calcul à partir des tables de la base de données.

On peut diviser toutes les opérations réalisables par un select en deux grandes catégories :

- Les opérations s'appliquant à une seule table
- Les opérations s'appliquant à plusieurs tables

# Les opérations s'appliquant à une seule table

Il y a 5 grandes catégories d'opérations s'appliquant à une seule table :

- Les filtres sur les colonnes = **Projection**. On choisit une ou plusieurs colonnes.
- La création d'attributs calculés = **Projection**. On crée une colonne par du calcul.
- Les filtres sur les lignes = **Restriction**. On choisit une ou plusieurs lignes.
- Les tris
- Les opérations statistiques

# Les opérations s'appliquant à plusieurs tables

Il y a 3 grandes catégories d'opérations s'appliquant à plusieurs tables :

- Les **opérations ensemblistes classiques** : union, intersection, différence qui travaillent plutôt sur des tuples de même nature (donc sur la même table).
- Le **produit cartésien** et la jointure associée qui permet de travailler sur deux colonnes de 2 tables différentes.
- Les imbrications d'opérations.



## Projection = filtre des colonnes

- **Projection = filtre des colonnes : tous les tuples, certains attributs**

*Donner la liste de tous les employés avec tous leurs attributs :*

```
SELECT * FROM emp;
```

C'est assez rare qu'on veuille tous les attributs. Il faut en tout cas se limiter à ce qui est nécessaire.

La projection d'une table est une nouvelle table constituée de tous les tuples et de certains attributs de la table de départ.

# Deux types de projection

Quand on crée une nouvelle table, en toute rigueur, il faut éviter qu'il y ait des tuples en double.

Il y a deux manières d'éviter les doublons :

- Projeter la clé primaire : on va parler de « **projection primaire** ».
- Eliminer les doublons : on va parler de « **projection avec distinct** ».

## La projection primaire

### Exemple

Tous les employés avec leurs fonctions.

```
SELECT nom, fonction
```

```
FROM emp;
```

Le résultat est une table d'employés avec moins d'attributs.

## Projection avec élimination des doublons : la clause **distinct**

### Exemple

La clause **distinct** permet, à partir d'une projection, d'éliminer les tuples en doubles

➤ Tous les métiers de la société :

```
SELECT distinct fonction
```

```
FROM emp ;
```

# Remarques sur la clé primaire

Si la liste d'attributs projetés contient la clé primaire, alors le distinct ne sert à rien:

```
SELECT distinct clé primaire, liste d'attributs
```

```
FROM table ;
```

équivalent à :

```
SELECT clé primaire, liste d'attributs FROM table ;
```

# Création d'attributs calculés

## Création d'un attribut

Il est possible de créer des attributs qui soient le résultat d'une opération arithmétique ou autre faite à partir d'autres attributs.

```
SELECT liste d'attributs avec des opérations sur attributs  
  
FROM table ;
```

➤ tous les salaires, commissions et salaires totaux des salariés :

```
SELECT nom, sal, comm, sal + comm  
  
FROM emp;
```

## Autres exemples

- *nombre de lettres du nom de chaque employé :*

```
SELECT NE, nom, length(nom)
FROM emp;
```

- *année d'embauche de chaque employé*

```
SELECT NE, nom, datemb, year(datemb)
FROM emp;
```

- *Salaire par tranche : petit (<1000), moyen (<2000), gros : autres.*

```
SELECT NE, nom,
CASE
  WHEN sal < 1000 THEN "petit"
  WHEN sal < 2000 THEN "moyen"
  ELSE "gros"
END
FROM emp;
```

# Renommer un attribut

## ➤ *Exemple de renommage*

```
SELECT NE, nom, sal, comm, sal + comm AS salaireTotal  
FROM emp;
```

On met « **AS** » entre l'ancien nom et le nouveau nom.

```
SELECT ancien_nom AS nouveau_nom FROM table;
```

Remarques

Si le nouveau nom contient des espaces, on écrira :

```
SELECT ancien_nom AS "nouveau nom" FROM table;
```

On peut se passer du AS :

```
SELECT ancien_nom nouveau_nom FROM table;
```



# Les opérations possibles pour calculer un attribut

- **Opérateurs et fonctions classiques**

- IF (pas standard)**

Le IF permet de faire des tests dans le SELECT avec 2 alternatives, 2 embranchements. Le passage par un embranchement permet de renvoyer une valeur et une seule : celle de l'attribut calculé.

Le IF est un cas particulier du CASE qui réduit les valeurs possibles à 2 :

IF(expr1,expr2,expr3) : retourne expr2 si expr1 est différent de 0 et de null, expr3 sinon.

# Les opérations possibles pour calculer un attribut

## ➤ Exemple 1 : IF simple

On crée un attribut, « categorie » qui aura deux valeurs : « bas » et « gros » en fonction du salaire.  
On met la limite à 2000.

Syntaxe : if(condition, résultat si vrai, résultat si faux)

```
SELECT NE, nom, sal,  
       IF(sal<2000,"1:bas","3:gros") AS categorie  
FROM emp  
ORDER BY categorie, nom;
```

## ➤ Exemple 2 : IF imbriqué

Pour avoir plus de deux alternatives, on peut imbriquer les IF

```
SELECT NE, nom, sal,  
       IF(sal<1200,"1:bas",IF(sal<2500,"2:moyen","3:gros"))  
       AS categorie  
FROM emp  
ORDER BY categorie, nom;
```

# Les opérations possibles pour calculer un attribut

## CASE WHEN : standard

### Présentation

Le “case when” est **une façon plus lisible d’écrire des IF imbriqués**.

En général, il a **deux usages et deux syntaxes** :

- usage « **else-if** » **classique** : succession de tests indépendants.
- usage « **switch** » **classique** : comparaison d’égalité entre une expression unique de départ.

Dans tous les cas, **le CASE WHEN renvoie une seule valeur** : c’est celle qui sera prise en compte dans la projection.

En général le ELSE final ou un DEFAULT permet de garantir que tous les cas possibles seront traités.

## *Première syntaxe : équivalent de IF imbriqués (else-if)*

- La première syntaxe permet de calculer le nouvel attribut à partir d'une succession de tests distincts les uns des autres. C'est un « else if » algorithmiq

### Syntaxe :

```
CASE
  WHEN expression opérateur valeur THEN résultat
  [* WHEN ... THEN ...]
  [ELSE résultat]
END
```

### Exemple :

```
SELECT NE, nom, sal,
CASE
  WHEN sal<1200 THEN "1:petit"
  WHEN sal<2500 THEN "2:moyen"
  ELSE "3:gros"
END AS categorie
FROM emp;
```

## *Deuxième syntaxe : switch classique*

- La deuxième syntaxe permet de calculer le nouvel attribut à partir d'une **succession de comparaisons d'égalité** entre une expression unique de départ et différentes valeur : c'est un « case » ou un « switch » algorithmique classique.

### Syntaxe :

```
CASE expression  
  WHEN valeur THEN résultat1  
  [* WHEN ... THEN ...]  
  [ELSE résultat]  
END
```

### Exemple :

```
SELECT NE, nom, sal,  
CASE sal div 1000  
  WHEN 0 THEN "< mille"  
  WHEN 1 THEN "mille et plus"  
  WHEN 2 THEN "2 mille et plus"  
  WHEN 3 THEN "3 mille et plus"  
  ELSE "plus de 4 mille"  
END AS categorie  
FROM emp;
```

# Select imbriqué dans la projection

On peut imbriquer un select dans les attributs projetés à condition que ce select ne renvoie qu'une seule valeur.

Exemple :

```
SELECT NE, nom, sal,
```

```
(SELECT max(sal)FROM emp) AS maxSal
```

```
FROM emp;
```

Usage :

**MIEUX VAUT EVITER LES SELECT IMBRIQUES !**