

# **Cours SGBD 1**

## **Concepts et langages des Bases de Données Relationnelles**

**SUPPORT DE COURS**

**IUT de Nice – Département INFORMATIQUE**

## **Plan**

<b>Chapitre 1</b>	<b>Introduction générale</b>
<b>Chapitre 2</b>	<b>Le modèle relationnel</b>
<b>Chapitre 3</b>	<b>Présentation des données</b>
<b>Chapitre 4</b>	<b>L'algèbre relationnelle</b>
<b>Chapitre 5</b>	<b>Le langage QBE</b>
<b>Chapitre 6</b>	<b>Le langage SQL</b>
<b>Chapitre 7</b>	<b>Gestion des transactions</b>
<b>Chapitre 8</b>	<b>Programmation avec VBA</b>
<b>Chapitre 9</b>	<b>Les objets dans Access</b>
<b>Chapitre 10</b>	<b>L'interface DAO</b>
<b>Chapitre 11</b>	<b>Le mode client serveur et ODBC</b>
<b>Chapitre 12</b>	<b>Automation et le modèle DCOM</b>

# **Chapitre 1      Introduction générale**

- I.      Notions intuitives**
- II.     Objectifs et avantages des SGBD**
- III.    L'architecture ANSI/SPARC**
- IV.    Notion de modélisation des données**
- V.     Survol des différents modèles de données**
- VI.    Bref historique,  
principaux SGBD commercialisés**

# I Notions intuitives

## . Base de données

ensemble structuré de données apparentées qui modélisent un univers réel

Une BD est faite pour enregistrer des faits, des opérations au sein d'un organisme  
(administration, banque, université, hôpital, ...)

Les BD ont une place essentielle dans l'informatique

## . Système de Gestion de Base de Données (SGBD)

**DATA BASE MANAGEMENT SYSTEM (DBMS)**

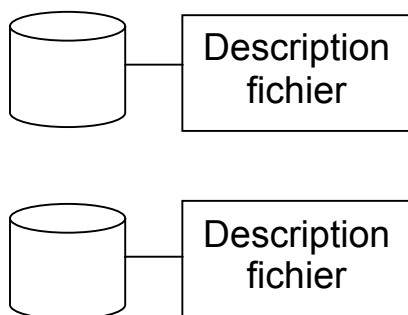
système qui permet de gérer une BD partagée par plusieurs utilisateurs simultanément

## . Des fichiers aux Base de Données

### Séparation des données et des programmes

#### FICHIER

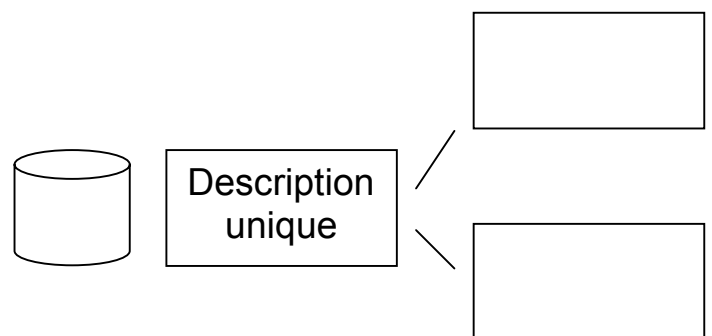
Les données des fichiers sont décrites dans les programmes



Programmes

#### BASE DE DONNEES

Les données de la BD sont décrites hors des programmes dans la base elle-même



Programmes

La multiplication des fichiers entraînait la *redondance* des données, ce qui rendait difficile les mises à jour.

D'où l'idée *d'intégration* et de *partage* des données

## II Objectifs et avantages des SGBD

Que doit permettre un SGBD ?

### □ Décrire les données

indépendamment des applications (de manière intrinsèque)

⇒ *langage de définition* des données

**DATA DEFINITION LANGUAGE (DDL)**

### □ Manipuler les données

interroger et mettre à jour les données  
sans préciser d'algorithme d'accès

dire QUOI sans dire COMMENT

langage de *requêtes* déclaratif

ex.:

quels sont les noms des produits de prix < 100F ?

⇒ *langage de manipulation* des données

**DATA MANIPULATION LANGUAGE (DML)**

## ▣ Contrôler les données

### *intégrité*

vérification de contraintes d'intégrité  
ex.: le salaire doit être compris entre 400F et 20000F

### *confidentialité*

contrôle des droits d'accès, autorisation

⇒ *langage de contrôle des données*

**DATA CONTROL LANGUAGE (DCL)**

## ▣ Partage

une BD est partagée entre plusieurs utilisateurs en même temps

⇒ contrôle des accès concurrents

notion de **transaction**

L'exécution d'une transaction doit préserver la cohérence de la BD

## ▣ Sécurité

reprise après panne, journalisation

## ▣ Performances d'accès

**index** (hashage, arbres balancés ...)



## ▣ Indépendance physique

Pouvoir modifier les structures de stockage ou les index sans que cela ait de répercussion au niveau des applications

Les disques, les méthodes d'accès, les modes de placement, le codage des données ne sont pas apparents

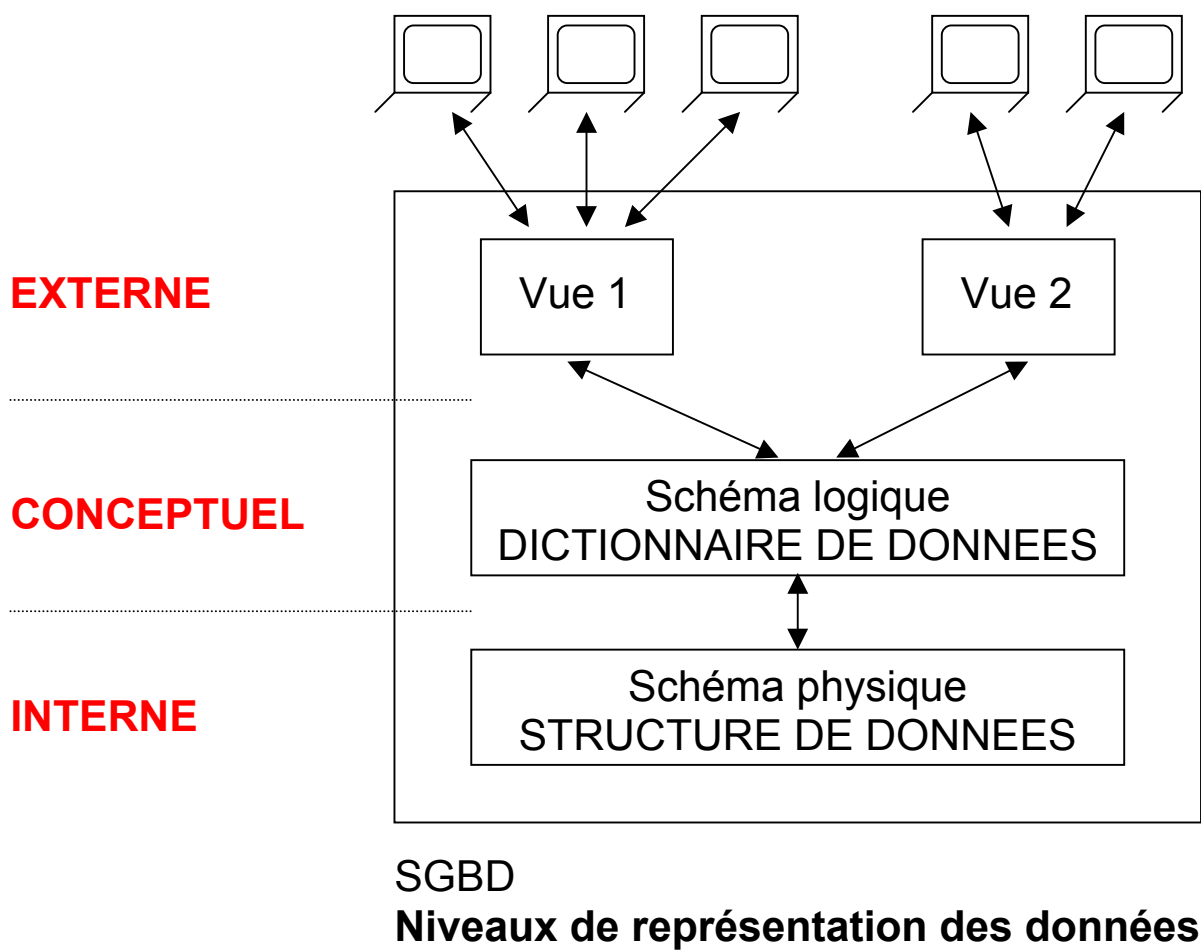
## ▣ Indépendance logique

Permettre aux différentes applications d'avoir des vues différentes des mêmes données

Permettre au DBA de modifier le schéma logique sans que cela ait de répercussion au niveau des applications

### III L'architecture ANSI/SPARC

- proposition en 75 de l' ANSI/SPARC  
(Standard Planning And Requirement Comitte)
- 3 niveaux de représentation des données



## ▣ Le niveau externe

Le concept de **vue** permet d'obtenir l'indépendance logique

La modification du schéma logique n'entraîne pas la modification des applications  
(une modification des vues est cependant nécessaire)

Chaque vue correspond à la perception d'une partie des données, mais aussi des données qui peuvent être synthétisées à partir des informations représentées dans la BD (par ex. statistiques)

## ▣ Le niveau conceptuel

il contient la description des données et des contraintes d'intégrité (Dictionnaire de Données)

le schéma logique découle d'une activité de modélisation

## ▣ Le niveau interne

il correspond aux structures de stockage et aux moyens d'accès (index)

**Pour résumer :**

## **Les fonctions des SGBD**

### **. DEFINITION DES DONNEES**

⇒ ***Langage de définition des données (DDL)***  
(conforme à un modèle de données)

### **. MANIPULATION DES DONNEES**

Interrogation

Mise à jour

insertion, suppression, modification

⇒ ***Langage de manipulation des données (DML)***  
(langage de requête déclaratif)

### **. CONTRÔLE DES DONNEES**

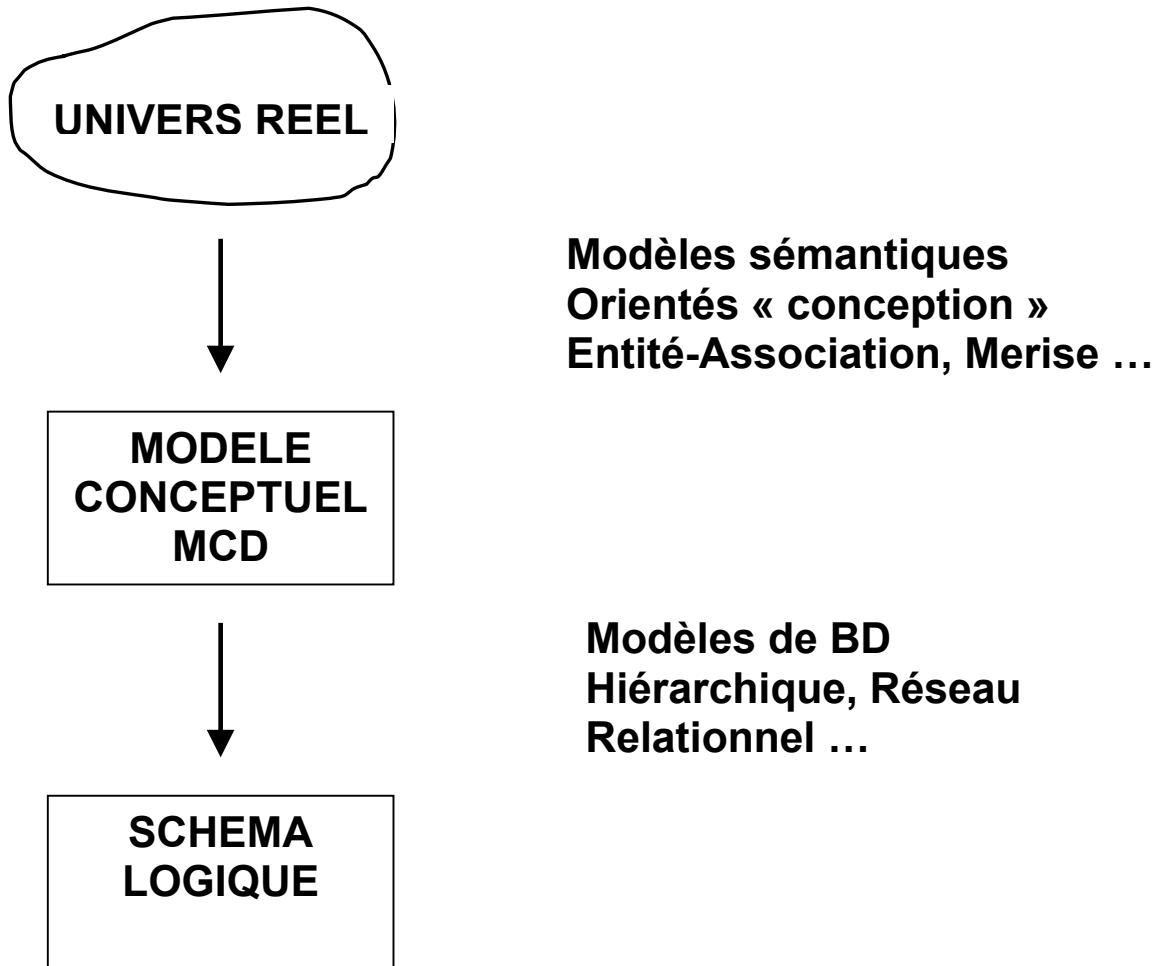
Contraintes d'intégrité

Contrôle des droits d'accès

Gestion de transactions

⇒ ***Langage de contrôle des données (DCL)***

## IV Notion de modélisation des données



- Les modèles de BD sont souvent trop limités pour pouvoir représenter directement le monde réel
- Méthodologies de conception présentées en ACSI, SGBD2

# Le modèle Entité-Association

EA en français, ER en anglais (pour Entity Relationship)

Formalisme retenu par l'ISO pour décrire l'aspect conceptuel des données à l'aide d'*entités* et d'*associations*

## ▣ Le concept d'entité

Représentation d'un objet matériel ou immatériel

Par exemple un employé, un projet, un bulletin de paie

Nom de l'entité
Liste des propriétés

- Les entités peuvent être regroupées en **types d'entités**

Par exemple, on peut considérer que tous les employés particuliers sont des **instances** du type d'entité générique EMPLOYE

Par exemple l'employé nommé DUPONT est une instance ou occurrence de l'entité EMPLOYE

## ▣ Les propriétés

données élémentaires relatives à une entité

Par exemple, un numéro d'employé, une date de début de projet

- on ne considère que les propriétés qui intéressent un contexte particulier
- Les propriétés d'une entité sont également appelées **des attributs**, ou des caractéristiques de cette entité

## ▣ L'identifiant

propriété ou groupe de propriétés qui sert à identifier une entité

L'identifiant d'une entité est choisi par l'analyste de façon à ce que deux occurrences de cette entité ne puissent pas avoir le même identifiant

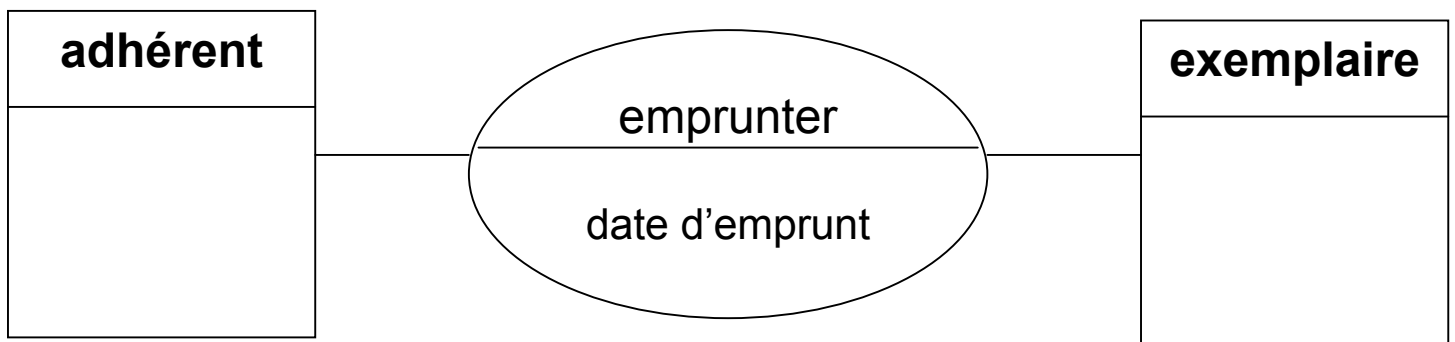
Par exemple, le numéro d'employé sera l'identifiant de l'entité EMPLOYE

## ▣ Les associations

Représentation d'un lien entre deux entités ou plus

- une association peut avoir des propriétés particulières

Par exemple, la date d'emprunt d'un livre



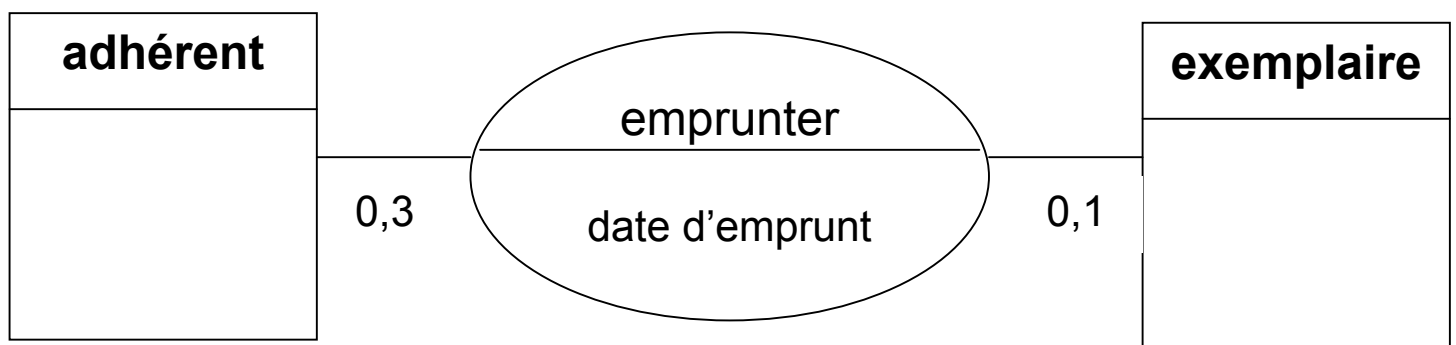


## ▣ Les cardinalités

La cardinalité d'une association pour une entité constituante est constituée d'une borne minimale et d'une borne maximale :

- Minimale : nombre minimum de fois qu'une occurrence de l'entité participe aux occurrences de l'association, généralement 0 ou 1
- Maximale : nombre maximum de fois qu'une occurrence de l'entité participe aux occurrences de l'association, généralement 1 ou n

Par exemple :



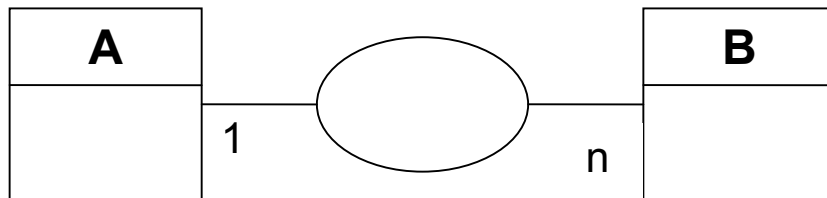
- La cardinalité 0,3 indique qu'un adhérent peut être associé à 0, 1, 2 ou 3 livres, c'est à dire qu'il peut emprunter au maximum 3 livres.
- A l'inverse un livre peut être emprunté par un seul adhérent, ou peut ne pas être emprunté.

- Les cardinalités maximum sont nécessaires pour concevoir le schéma de la base de données
- Les cardinalités minimums sont nécessaires pour exprimer les contraintes d'intégrité

En notant uniquement les cardinalités maximum, on distingue 3 type de liens :

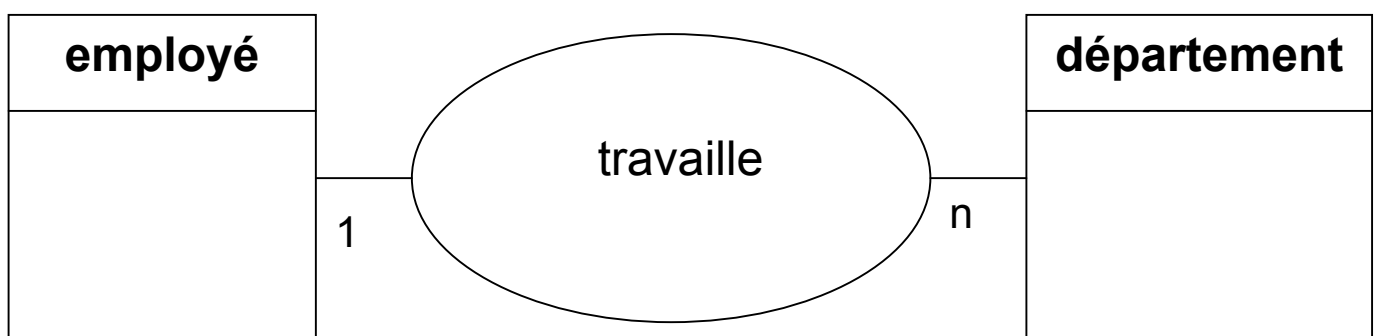
- **Lien fonctionnel 1:n**
- **Lien hiérarchique n:1**
- **Lien maillé n:m**

## Lien fonctionnel 1:n



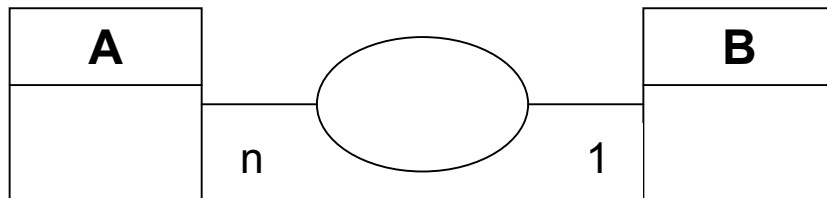
Une instance de A ne peut être associée qu'à une seule instance de B

Par exemple :



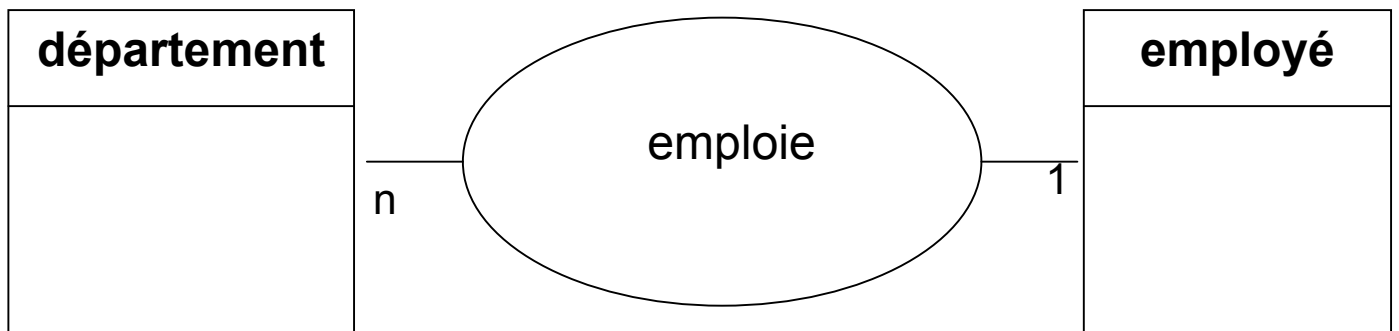
Un employé ne peut travailler que dans un seul département

## Lien hiérarchique n:1



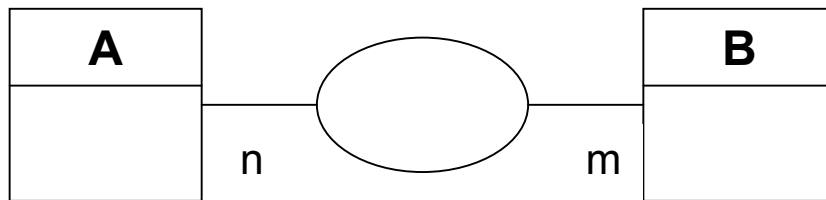
Une instance de A peut être associée à plusieurs instances de B

Inverse d'un lien 1:n



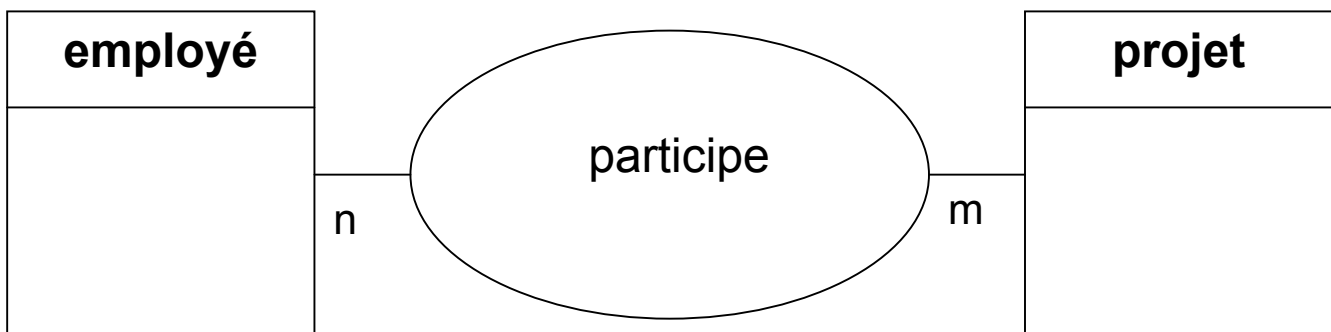
Un département emploie généralement plusieurs employés

## Lien maillé n:m



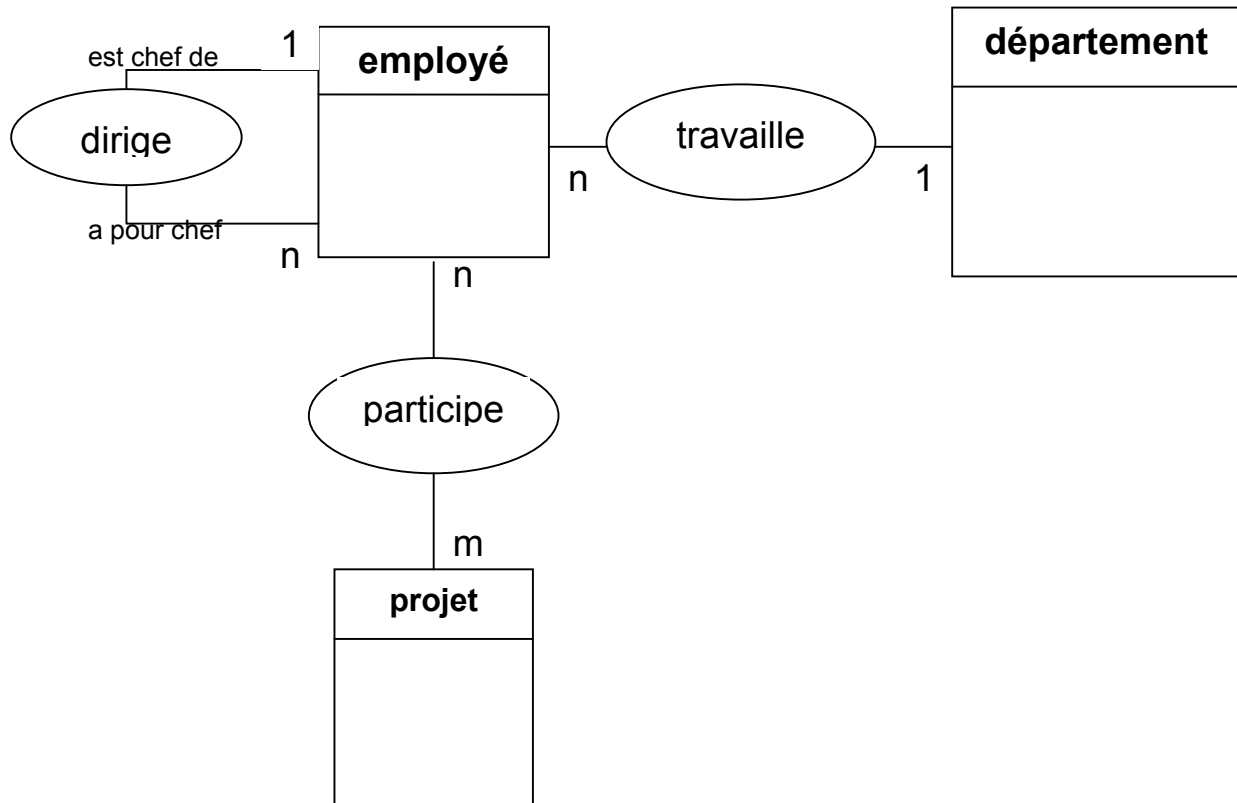
Une instance de A peut être associée à plusieurs instances de B et inversement

Par exemple :



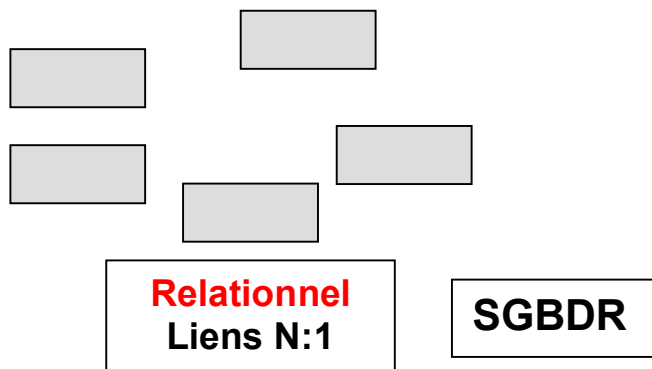
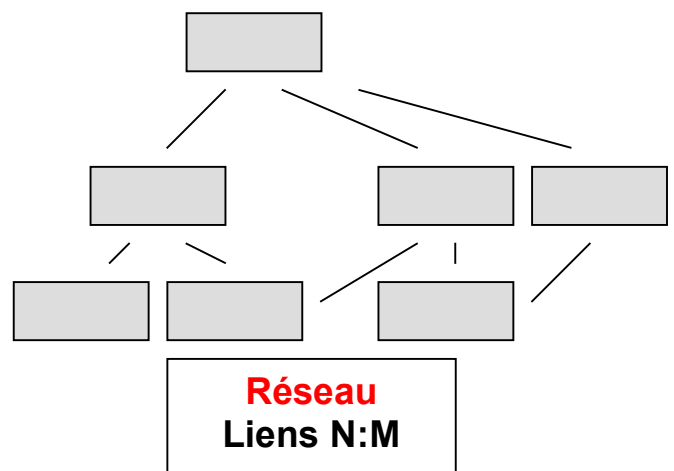
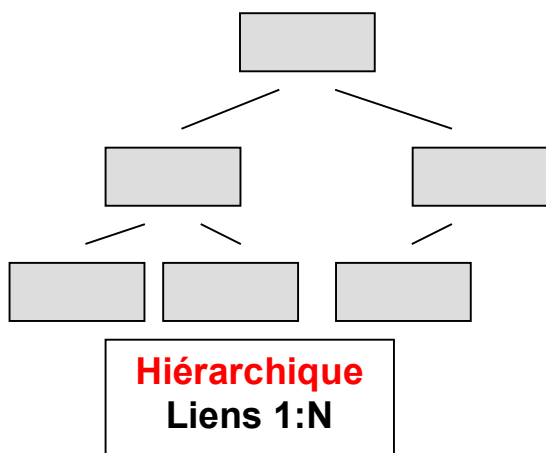
De ce schéma, on déduit qu'un employé peut participer à plusieurs projets.

## Exemple de diagramme Entité Association



## V Les différents modèles de données

- L'organisation des données au sein d'une BD a une importance essentielle pour faciliter l'accès et la mise à jour des données



**SGBDR**

- Les modèles hiérarchique et réseau sont issus du modèle **GRAPHE**
  - données organisées sous forme de graphe
  - langages d'accès navigationnels (adressage par liens de chaînage)
  - on les appelle "modèles d'accès"
- Le modèle relationnel est fondé sur la notion mathématique de **RELATION**
  - introduit par Codd (recherche IBM)
  - données organisées en tables (adressage relatif)
  - stratégie d'accès déterminée par le SGBD



## LE MODÈLE RÉSEAU

- Schéma logique représenté par un **GRAPHE**

noeud : article (représente une entité)

arc : lien hiérarchique 1:N

- Exemple de schéma réseau

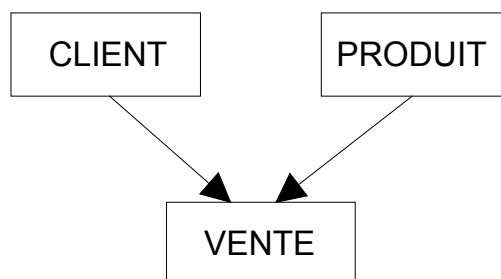
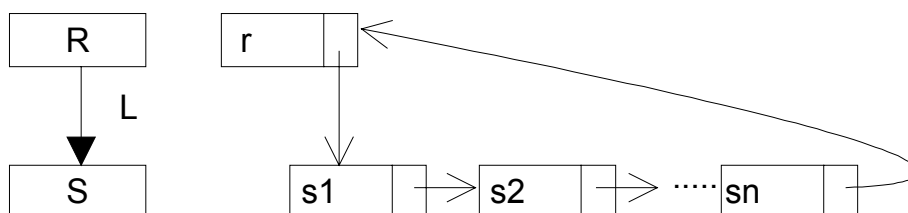


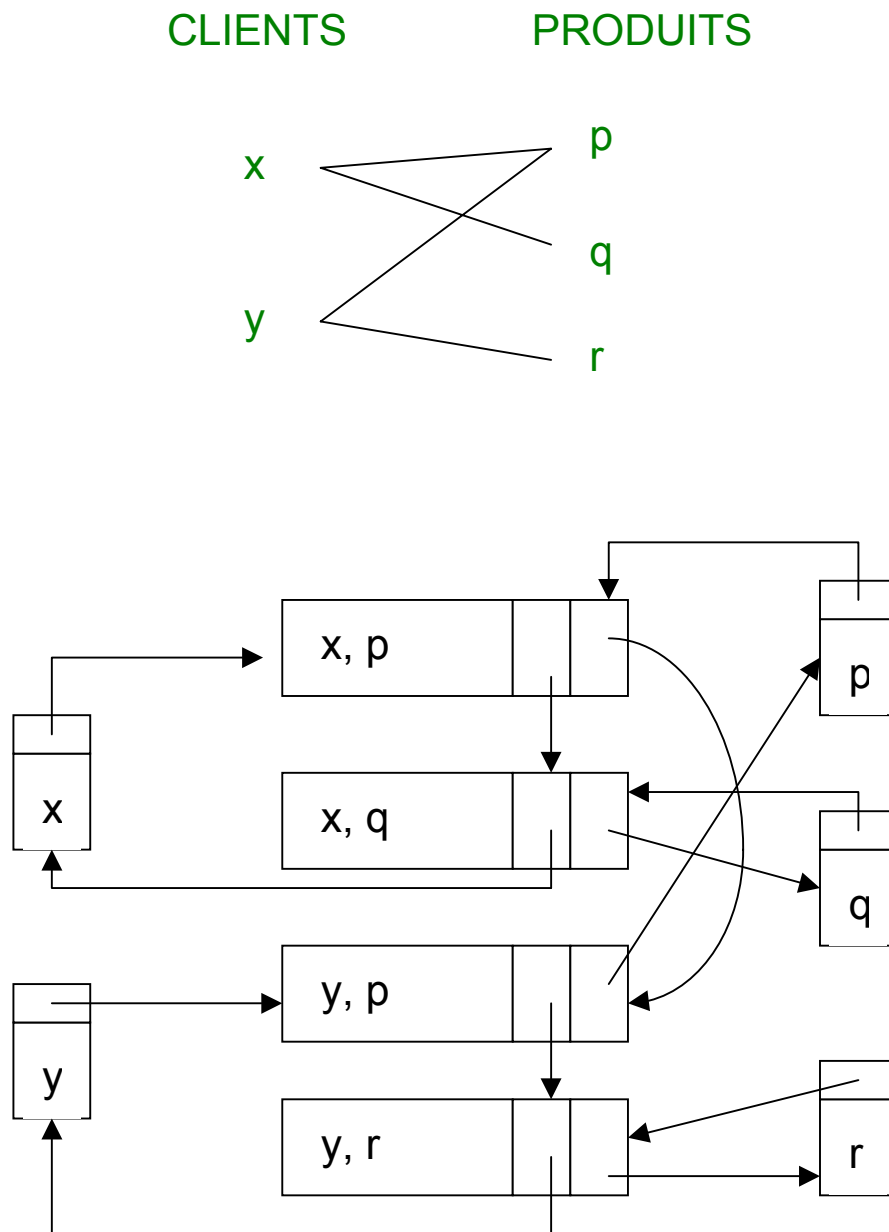
Diagramme de Bachman

- Langage navigationnel pour manipuler les données

- Implémentation d'un lien par une liste circulaire :



- Exemple de schéma réseau :



Représentation d'une association N:M par 2 liens  
CODASYL

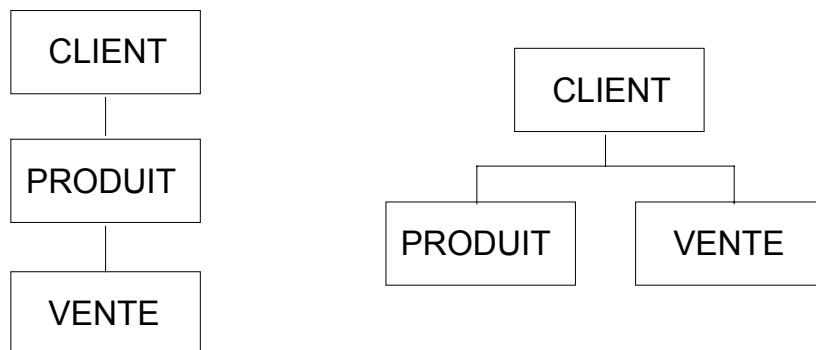
## LE MODÈLE HIÉRARCHIQUE

- Schéma logique représenté par un **ARBRE**

noeud : segment (regroupement de données)

arc : lien hiérarchique 1:N

- Exemple de schéma hiérarchique



- Choix possible entre plusieurs arborescences  
(le segment racine est choisi en fonction de l'accès souhaité)
- Dissymétrie de traitement pour des requêtes symétriques  
En prenant l'ex. précédent, considérer les 2 requêtes :
  - a) Trouver les no de produits achetés par le client x
  - b) Trouver les no de clients qui ont acheté le produit pElles sont traitées différemment suivant le choix du segment racine (Client ou Produit)
- Adéquation du modèle pour décrire des organisations à structure arborescente (ce qui est fréquent en gestion)

## LE MODÈLE RELATIONNEL

- En 1970, CODD présente le modèle relationnel
- Schéma logique représenté par des **RELATIONS**

### LE SCHÉMA RELATIONNEL

Le schéma relationnel est l'ensemble des **RELATIONS** qui modélisent le monde réel

- Les relations représentent les **entités** du monde réel (comme des personnes, des objets, etc.) ou les **associations** entre ces entités
- Passage d'un schéma conceptuel E-A à un schéma relationnel
  - une entité est représentée par la relation :  
**nom\_de\_l'entité (liste des attributs de l'entité)**
  - une association M:N est représentée par la relation :  
**nom\_de\_l'association (**  
**liste des identifiants des entités participantes,**  
**liste des attributs de l'association)**

- Ex . :

CLIENT (IdCli, nom, ville)

PRODUIT (IdPro, nom, prix, qstock)

VENTE (IdCli, IdPro, date, qte)

Représentation des données sous forme de **tables** :

CLIENT	IdCli	Nom	Ville
	X	Smith	Paris
	Y	Jones	Paris
	Z	Blake	Nice

PRODUIT	IdPro	Nom	Prix	Qstock
	P	Auto	100	10
	Q	Moto	100	10
	R	Velo	100	10
	S	Pedalo	100	10

VENTE	IdCli	IdPro	Date	Qte
	X	P		1
	X	Q		2
	X	R		3
	Y	P		4
	Y	Q		5
	Z	Q		6

## LES AVANTAGES DU MODÈLE RELATIONNEL

- **SIMPLICITE DE PRÉSENTATION**

- représentation sous forme de **tables**

- **OPÉRATIONS RELATIONNELLES**

- algèbre relationnelle
- langages assertionnels

- **INDEPENDANCE PHYSIQUE**

- optimisation des accès
- stratégie d'accès déterminée par le système

- **INDEPENDANCE LOGIQUE**

- concept de **VUES**

- **MAINTIEN DE L'INTEGRITÉ**

- contraintes d'intégrité définies au niveau du schéma

## **VI Bref historique, principaux systèmes**

### **Années 60 Premiers développements des BD**

- fichiers reliés par des pointeurs
- systèmes IDS 1 et IMS 1 précurseurs des SGBD modernes

### **Années 70 Première génération de SGBD**

- apparition des premiers SGBD
- séparation de la description des données de la manipulation de celles-ci par les applications
- modèles hiérarchique et réseau CODASYL
- langages d'accès navigationnels
- SGBD IDMS, IDS 2 et IMS 2

### **Années 80 Deuxième génération**

- modèle relationnel
- les SGBDR représentent l'essentiel du marché BD (aujourd'hui)
- architecture répartie client-serveur

### **Années 90 Troisième génération**

- modèles de données plus riches
- systèmes à objets  
OBJECTSTORE, O2

## **Principaux systèmes**

- Oracle
- DB2 (IBM)
- Ingres
- Informix
- Sybase
- SQL Server (Microsoft)
- O2
- Gemstone

## **Sur micro :**

- Access
- Paradox
- FoxPro
- 4D
- Windev

## **Sharewares :**

- MySQL
- MSQL
- Postgres
- InstantDB



## **Chapitre 2      Le modèle relationnel**

### **I. LES CONCEPTS**

### **II. LES DÉPENDANCES FONCTIONNELLES**

### **III. LES RÈGLES D'INTÉGRITÉ**

### **IV. LES FORMES NORMALES**

# **I    LES CONCEPTS**

- **LE DOMAINE**
- **LA RELATION**
- **LES N-UPLETS**
- **LES ATTRIBUTS**
- **LE SCHÉMA D'UNE RELATION**
- **LE SCHÉMA D'UNE BDR**
- **LA REPRÉSENTATION**

## ▣ LE DOMAINE

ensemble de valeurs atomiques d'un certain type sémantique

Ex. :

`NOM_VILLE = { Nice, Paris, Rome }`

- les domaines sont les ensembles de valeurs possibles dans lesquels sont puisées les données
- deux ensembles peuvent avoir les mêmes valeurs bien que sémantiquement distincts

Ex. :

`NUM_ELV = { 1, 2, ... , 2000 }`

`NUM_ANNEE = { 1, 2, ... , 2000 }`

## ▣ LA RELATION

sous ensemble du produit cartésien de plusieurs domaines

$$R \subset D1 \times D2 \times \dots \times Dn$$

$D1, D2, \dots, Dn$  sont les domaines de  $R$   
 $n$  est le degré ou l'arité de  $R$

Ex.:

Les domaines :

$NOM\_ELV = \{ \text{dupont, durant} \}$

$PREN\_ELV = \{ \text{pierre, paul, jacques} \}$

$DATE\_NAISS = \{ \text{Date entre 1/1/1990 et 31/12/2020} \}$

$NOM\_SPORT = \{ \text{judo, tennis, foot} \}$

La relation ELEVE

$ELEVE \subset NOM\_ELV \times PREN\_ELV \times DATE\_NAISS$

$ELEVE = \{ (\text{dupont, pierre, 1/1/1992}),$   
 $\qquad\qquad\qquad (\text{durant, jacques, 2/2/1994}) \}$

La relation INSCRIPT

$INSCRIPT \subset NOM\_ELV \times NOM\_SPORT$

$INSCRIPT = \{ (\text{dupont, judo}), (\text{dupont, foot}),$   
 $\qquad\qquad\qquad (\text{durant, judo}) \}$

## ▣ LES N-UPLETS

un élément d'une relation est un n-uplet de valeurs  
(tuple en anglais)

- un **n-uplet** représente un fait

Ex.:

« Dupont pierre est un élève né le 1 janvier1992 »

« dupont est inscrit au judo »

- **DEFINITION PRÉDICATIVE D'UNE RELATION**

Une relation peut être considérée comme un **PRÉDICAT**  
à n variables

$$\theta(x, y, z) \text{ vrai} \Leftrightarrow (x, y, z) \in R$$

Ex. :

$$\text{est\_inscrit}(\text{dupont}, \text{judo}) \Leftrightarrow (\text{dupont}, \text{judo}) \in \text{INSCRIPT}$$

## ▣ LES ATTRIBUTS

Chaque composante d'une relation est un attribut

- Le nom donné à un attribut est porteur de sens
- Il est en général différent du nom de domaine
- Plusieurs attributs peuvent avoir le même domaine

Ex. :

La relation **TRAJET** :

$$\text{TRAJET} \subset \text{NOM\_VILLE} \times \text{NOM\_VILLE}$$

Dans laquelle la première composante représente la ville de départ **VD**, la deuxième composante la ville d'arrivée **VA** d'un trajet.

## ▣ LE SCHÉMA D'UNE RELATION

Le schéma d'une relation est défini par :

- le nom de la relation
- la liste de ses attributs

on note :  $R (A1, A2, \dots, An)$

Ex.:

ELEVE (NOM, PRENOM, NAISS)  
INSCRIPT (NOM\_ELIV, SPORT)  
TRAJET (VD, VA)

### • Extension et Intension

- **L'extension** d'une relation correspond à l'ensemble de ses éléments (n-uplets)

→ le terme **RELATION** désigne une extension

- **L'intention** d'une relation correspond à sa signification

→ le terme **SCHÉMA DE RELATION** désigne l'intention d'une relation

## ▣ LE SCHÉMA D'UNE BDR

Le schéma d'une base de données est défini par :

- l'ensemble des schémas des relations qui la composent

Notez la différence entre :

- le schéma de la BDR qui dit comment les données sont organisées dans la base
- l'ensemble des n-uplets de chaque relation, qui représentent les données stockées dans la base

### • Conception de Schéma Relationnel

- Problème :  
Comment choisir un schéma approprié ?

- Méthodologies de conception

→ cours ACSI

→ cours SGBD 2



## ▣ LA REPRÉSENTATION

1 **RELATION** = 1 TABLE

U1	V1	W1	X1	Y1
U2	V2	W2	X2	Y2
U3	V3	W3	X3	Y3

1 **ÉLÉMENT** ou **n-uplet** = 1 LIGNE

LIGNE →  
1 élément

U1	V1	W1	X1	Y1

\* une relation est un ensemble  $\Rightarrow$  on ne peut pas avoir 2 lignes identiques

1 **ATTRIBUT** = 1 COLONNE

U1				
U2				
U3				



COLONNE

1 attribut ou propriété

Exemples :

- La relation ELEVE

ELEVE :	NOM	PRENOM	NAISS
élément →	dupont	Pierre	1/1/1992
	durant	Jacques	2/2/1994
	duval	Paul	3/03/81

- La relation INSCRIPT

INSCRIPT :	NOM_ELV	SPORT
élément →	Dupont	judo
	Dupont	foot
	Durant	judo

- La relation TRAJET

TRAJET :	VD	VA
élément →	Nice	paris
	Paris	rome
	Rome	nice

## Fenêtre Création de Table d'Access

Nom du champ	Type de données	Description
IdCli	Texte	numéro client
Nom	Texte	nom client
Ville	Texte	ville client

Propriétés du champ

Général    Liste de choix

Taille du champ: 5

Format:

Masque de saisie:

Légende:

Valeur par défaut:

Valide si:

Message si erreur:

Null interdit: Non

Chaîne vide autorisée: Non

Indexé: Oui - Sans doublons

Un nom de champ peut compter jusqu'à 64 caractères, espaces inclus. Pour obtenir de l'aide, appuyez sur F1.

## Affichage d'une table dans Access

Sélecteur d'enregistrement

	IdCli	Nom	Ville
▶	10	toto	Nice
▶	20	tata	Nice
▶	30	titi	Paris
▶	40	tutu	Rome
✱			Nice

Enr: 2 sur 4

Boutons de déplacement

## II 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.:

PRODUIT (no\_prod, nom, prixUHT)  
 $\text{no\_prod} \rightarrow (\text{nom}, \text{prixUHT})$

NOTE (no\_contrôle, no\_élève, note)  
 $(\text{no\_contrôle}, \text{no\_élève}) \rightarrow \text{note}$

- une dépendance fonctionnelle est une propriété **sémantique**, elle correspond à une contrainte supposée toujours vrai du monde réel

### D.F. élémentaire

D.F.  $X \rightarrow A$  mais  $A$  est un attribut unique non inclus dans  $X$  et il n'existe pas de  $X'$  inclus dans  $X$  tel que  $X' \rightarrow A$

## ▣ La clé d'une relation

attribut (ou groupe minimum d'attributs) qui détermine tous les autres

Ex.:

PRODUIT (no\_prod, nom, prixUHT)

no\_prod → (nom, prixUHT)

no\_prod est une clé

- Une clé détermine un n-uplet de façon unique
- Pour trouver la clé d'une relation, il faut examiner attentivement les hypothèses sur le monde réel
- Une relation peut posséder plusieurs clés, on les appelle clés candidates

Ex.:

dans la relation PRODUIT, nom est une clé candidate (à condition qu'il n'y ait jamais 2 produits de même nom)

## ▣ Clé primaire

choix d'une clé parmi les clés candidates

## ▣ Clé étrangère ou clé secondaire

attribut (ou groupe d'attributs) qui fait référence à la clé primaire d'une autre relation

Ex.:

CATEG (**no\_cat**, design, tva)

PRODUIT(no\_prod, nom, marque, no\_cat, prixUHT)

no\_cat dans PRODUIT est une clé étrangère

**CLÉ ÉTRANGÈRE** = CLÉ PRIMAIRE dans une autre relation

### **III LES RÈGLES D'INTÉGRITÉ**

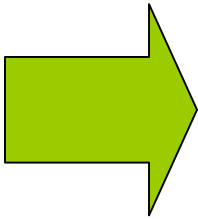
Les règles d'intégrité sont les assertions qui doivent être vérifiées par les données contenues dans une base

Le modèle relationnel impose les contraintes structurelles suivantes :

- ▣ **INTÉGRITÉ DE DOMAINE**
  - ▣ **INTÉGRITÉ DE CLÉ**
  - ▣ **INTÉGRITÉ RÉFÉRENCIELLE**
- La gestion automatique des contraintes d'intégrité est l'un des outils les plus importants d'une base de données.
  - Elle justifie à elle seule l'usage d'un SGBD.

## ▣ INTÉGRITÉ DE DOMAINE

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



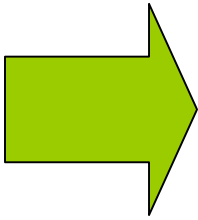
- contrôle des valeurs des attributs
- contrôle entre valeurs des attributs



## □ INTÉGRITÉ DE CLÉ

Les valeurs de clés primaires doivent être :

- uniques
- non NULL



- Unicité de clé
- Unicité des n-uplets

- Valeur NULL

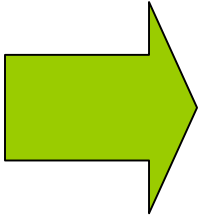
valeur conventionnelle pour représenter une information **inconnue**

- 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 RÉFÉRENCIELLE

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

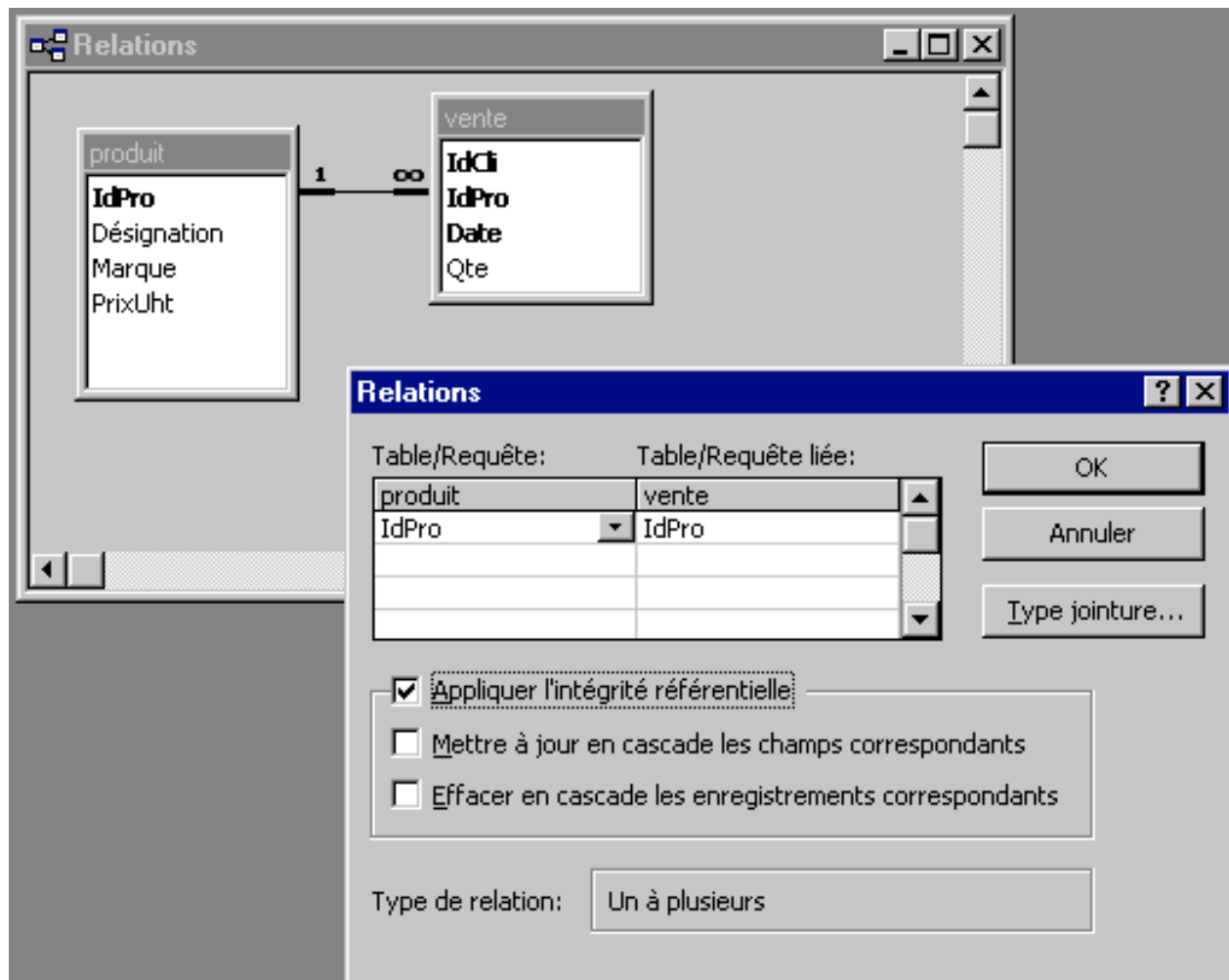
Exemple :

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

## Paramétrage des Relations dans Access



- IdPro de Vente est une clé étrangère qui fait référence à la clé primaire de Produit
- Appliquer l'intégrité référentielle signifie que l'on ne pourra pas avoir, à aucun moment, une ligne de Vente avec un code produit IdPro inexistant dans la table Produit.
- Une valeur de clé étrangère peut être Null

## IV LES FORMES NORMALES

### ▣ La théorie de la normalisation

- elle met en évidence les relations "indésirables"
- elle définit les critères des relations "désirables" appelées **formes normales**
- Propriétés indésirables des relations
  - Redondances
  - Valeurs NULL
- elle définit le processus de normalisation permettant de **décomposer** une relation non normalisée en un ensemble équivalent de relations normalisées

## ▣ La décomposition

### Objectif:

- décomposer les relations du schéma relationnel sans perte d'informations
  - obtenir des relations canoniques ou de base du monde réel
  - 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

**5 FN**, les critères sont de plus en plus restrictifs

$$FN_j \Rightarrow FN_i \quad (j > i)$$

- **Notion intuitive de FN**

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

## □ 1<sup>ère</sup> Forme Normale 1FN

Une relation est en 1FN si tout attribut est atomique (non décomposable)

### 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)

## □ 2<sup>ème</sup> 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
- 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**)



### □ 3<sup>ème</sup> 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)

### □ 3<sup>ème</sup> Forme Normale de BOYCE-CODD BCNF

Une relation est en BCNF :

- elle est en 1FN et
  - ssi les seules D.F. élémentaires sont celles dans lesquelles une clé détermine un attribut
- 
- BCNF signifie que l'on ne peut pas avoir un attribut (ou groupe d'attributs) déterminant un autre attribut et distinct de la clé
  - Ceci évite les redondances dans l'extension de la relation: mêmes valeurs pour certains attributs de n-uplets différents
  - BCNF est plus fin que FN3 : **BCNF  $\Rightarrow$  FN3**

### Contre-exemple

une relation en 3FN qui n'est pas BCNF

**CODEPOSTAL (ville, rue, code)**

on vérifie qu'elle est FN3, elle n'est pas BCNF car la clé = (ville, rue) (ou (code, ville) ou (code, rue)), et code  $\rightarrow$  ville

## **Chapitre 3      Présentation des données**

Une fois la base et les tables créées, il faut pouvoir les exploiter.

L'utilisateur final aura besoin de visualiser et saisir des données, d'effectuer des calculs et d'imprimer des résultats.

La réponse à ces problèmes de présentation des données est fournie par :

- **les formulaires**

destinés à être affichés à l'écran

- **les états**

destinés à être imprimés.

## I Les formulaires

2 types de formulaires :

- **de présentation des données**

Ils permettent de saisir, ou modifier les données d'une ou plusieurs tables sous une forme visuellement agréable

- **de distribution**

ils ne sont attachés à aucune table, et servent uniquement de page de menu pour orienter l'utilisateur vers d'autres formulaires ou états

## Formulaire rudimentaire

A screenshot of a simple form window titled "produit". The form contains four input fields: "IdPro" with the value "100", "Désignation" with the value "ps", "Marque" with the value "ibm", and "PrixUht" with the value "5 000,00 F". At the bottom, there is a navigation bar with the text "Enr: 1 sur 4" and several navigation buttons (back, forward, search, etc.).

## Fenêtre Conception de Formulaire d'Access

A screenshot of the Access Form Design window. The window is titled "Formulaire2 : Formulaire". It shows a design grid with three sections: "En-tête de formulaire" (Header), "Détail" (Detail), and "Pied de formulaire" (Footer). The "Détail" section is currently selected and contains a grid of 10 columns and 2 rows. A vertical toolbar on the left side of the window contains various design tools for creating and modifying form elements.

## Formulaire avec sous-formulaire

The screenshot shows a software window titled 'categorie'. It contains a main form with fields for 'IdCat', 'Désignation' (containing 'Ordinateurs'), and 'TauxTva' (containing '10'). Below these is a sub-form titled 'produit' which contains a table. The table has columns: 'IdPro', 'IdC', 'Désignation', 'Marque', 'PrixUht', and 'Qstoc'. It displays four rows of data for products related to 'Ordinateurs'. Below the table is a navigation bar for the sub-form with 'Enr: 1 sur 4'. At the bottom of the main window is another navigation bar for the main form with 'Enr: 1 sur 3'.

	IdPro	IdC	Désignation	Marque	PrixUht	Qstoc
▶	10	C1	ps	ibm	5 000,00 F	20
	20	C1	mac	apple	7 500,00 F	20
	30	C1	aptiva	ibm	12 000,00 F	10
	40	C1	power mac	apple	20 000,00 F	10
*		C1			0,00 F	0

Permet d'afficher les données de deux tables qui sont en relation l'une avec l'autre.

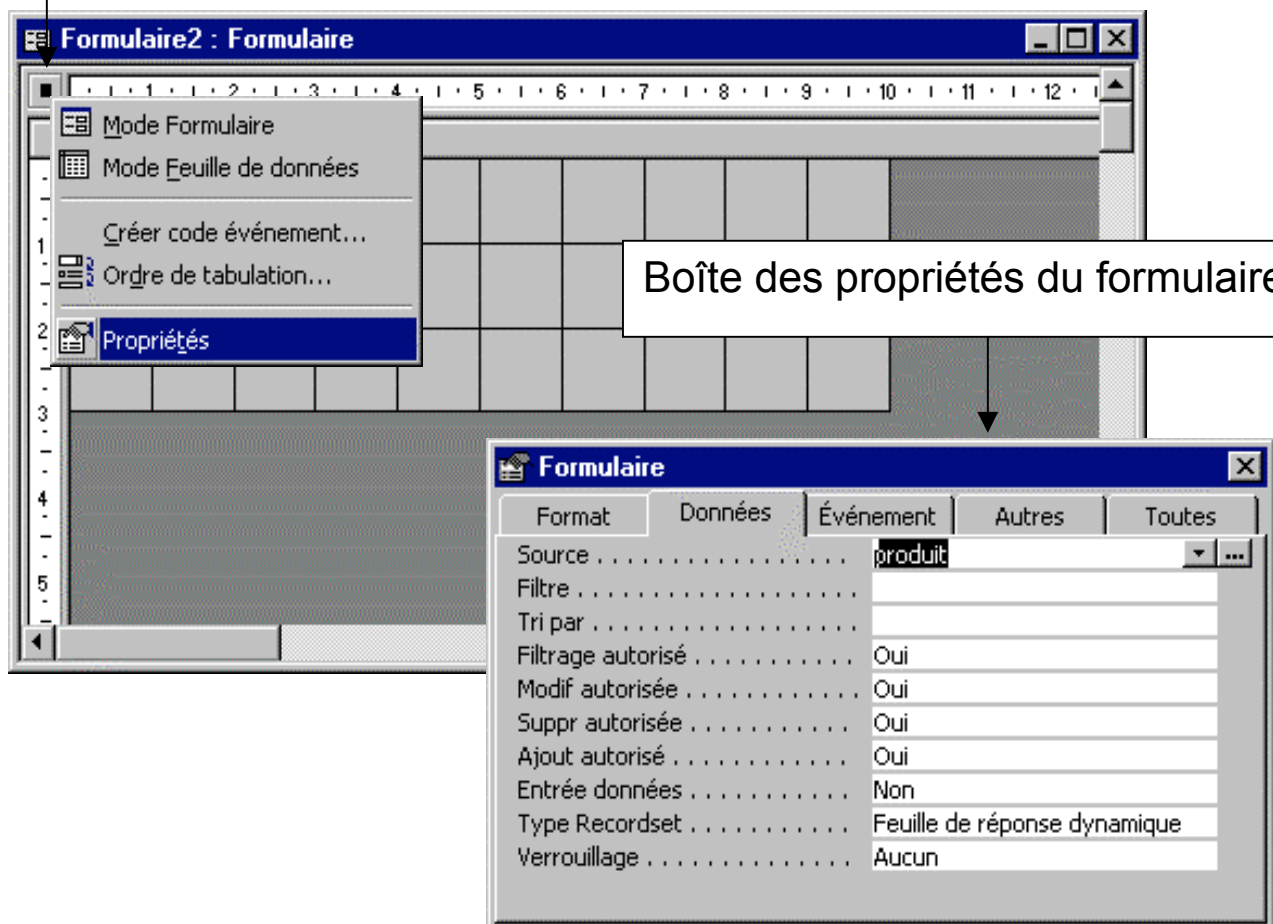
- Le formulaire principal affiche les données de la table principale
- Le sous formulaire affiche les données de la table liée

Si l'utilisateur change d'enregistrement principal, le sous formulaire est automatiquement mis à jour.

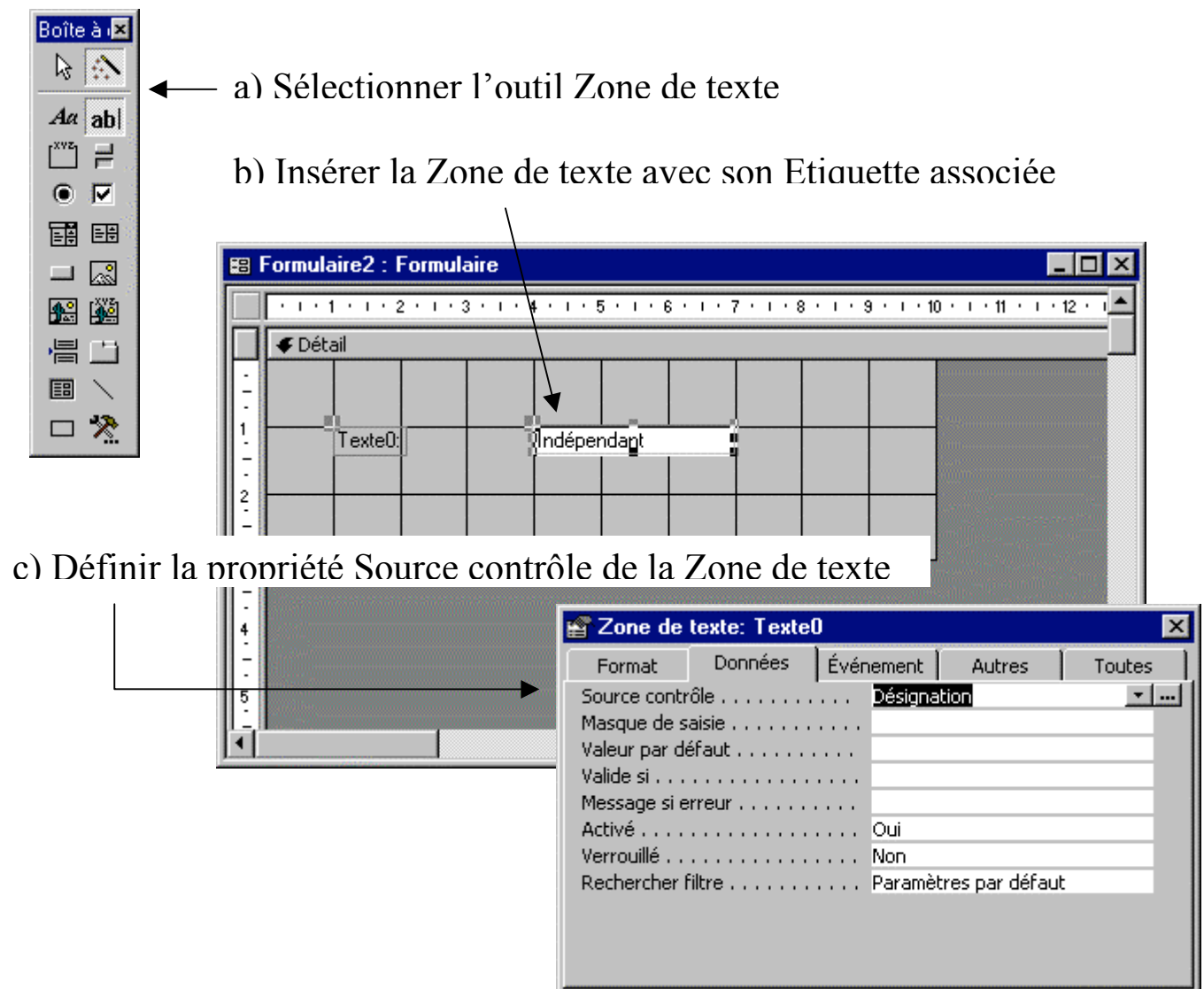
## Création d'un formulaire de présentation

### 1) Définir la propriété Source de données (table ou requête)

Cliquer ici avec le bouton droit, puis sélectionner Propriétés



## 2) Insérer dans le formulaire les Zones de texte liées aux champs de la Source de données



Pour afficher la fenêtre des propriétés d'un contrôle, cliquer dessus avec le bouton droit de la souris



## II Les états

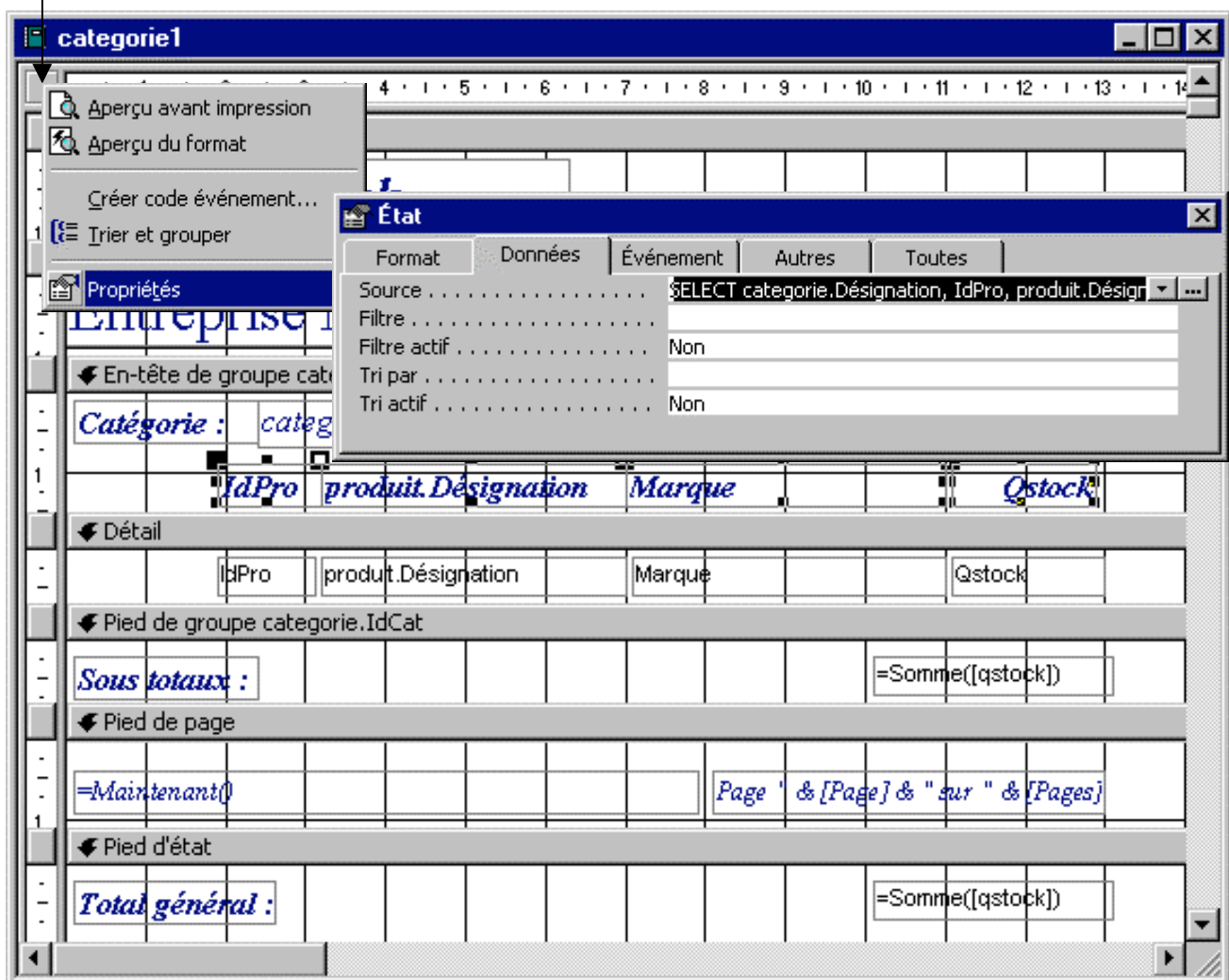
Un état permet d'imprimer des enregistrements, en les groupant et en effectuant des totaux et des sous totaux.

En-tête d'état	→	<i>Etat du Stock</i>			
En-tête de page	→	Entreprise MICRO			
En-tête de groupe	→	Catégorie :		O	
		<b>IdPro</b>	<b>Désignation</b>	<b>Marque</b>	<b>Qstock</b>
Détail	→	10	Ps	Ibm	10
		20	Imac	Apple	20
		30	Aptiva	Ibm	10
Pied de groupe	→	Sous totaux :			40
		...			
Pied de page	→	Jeudi 12 février 1998			Page 1 sur 1
Pied d'état	→	Total général :			200

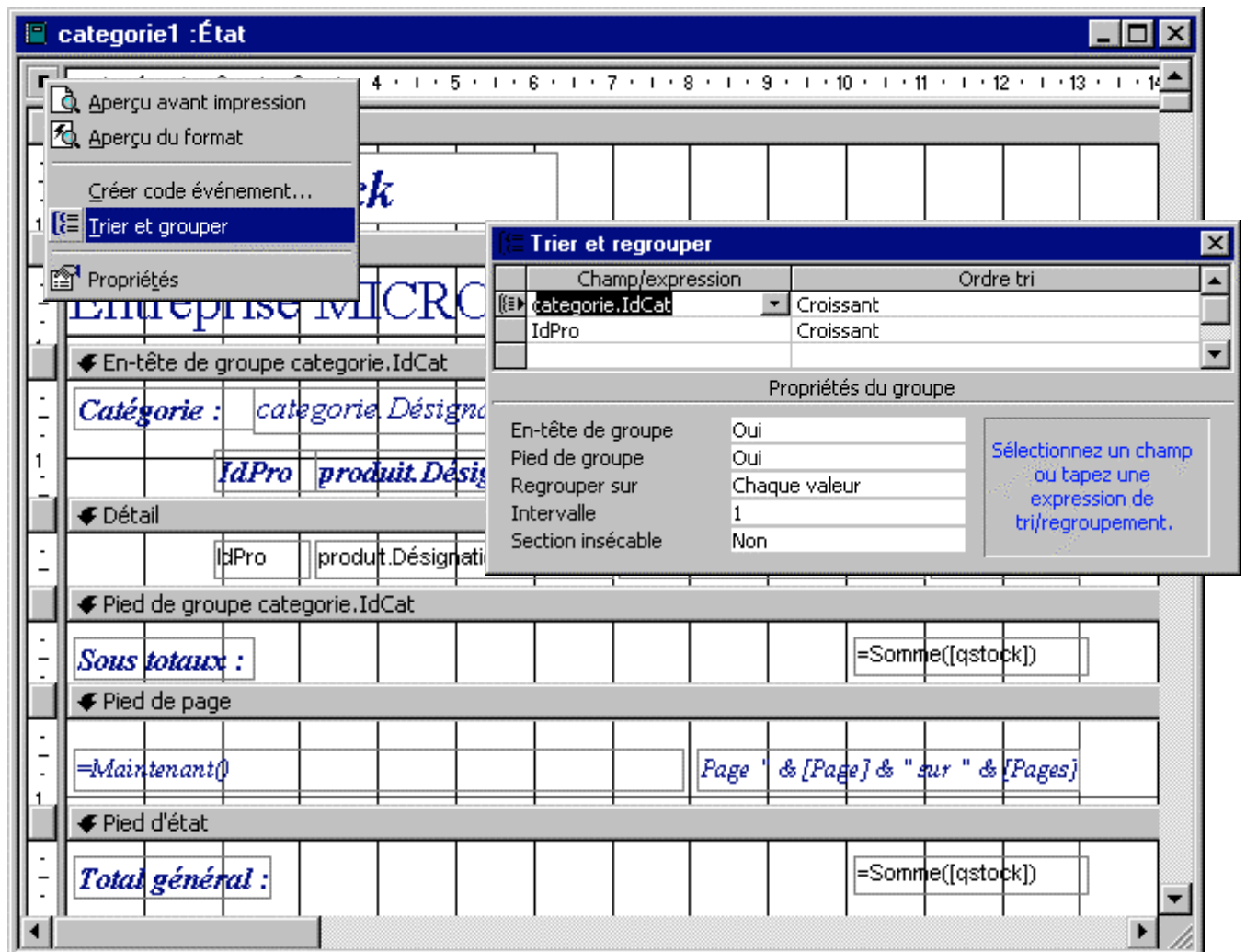
## Création d'un état

- 1) Définir la propriété Source de données (table ou requête)

Cliquer ici avec le bouton droit, puis sélectionner Propriétés



## 2) Définir Trier et grouper



### 3) Placer les champs dans les différentes section de l'état

category1 :État

1 2 3 4 5 6 7 8 9 10 11 12 13 14

En-tête d'état

*Etat du Stock*

En-tête de page

Entreprise MICRO

En-tête de groupe categorie.IdCat

Catégorie : categorie Désignation

IdPro produit.Désignation Marque Qstock

Détail

IdPro produit.Désignation Marque Qstock

Pied de groupe categorie.IdCat

Sous totaux : =Somme([qstock])

Pied de page

=Maintenant() Page ' & [Page] & " sur " & [Pages]

Pied d'état

Total général : =Somme([qstock])

## **Chapitre 4      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

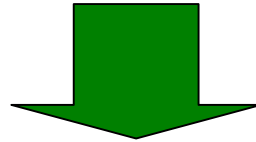
L'union de deux relations R1 et R2 de même schéma est une relation R3 de schéma identique qui a pour n-uplets les n-uplets de R1 et/ou R2

On notera :

$$\mathbf{R3 = R1 \cup R2}$$

R1			R2	
A	B	∪	A	B
0	1		0	1
2	3		4	5

$$\mathbf{R3 = R1 \cup R2}$$



R3	
A	B
0	1
2	3
4	5

## □ INTERSECTION

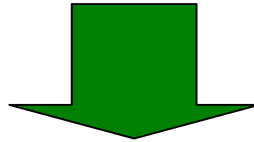
L'intersection entre deux relations R1 et R2 de même schéma est une relation R3 de schéma identique ayant pour n-uplets les n-uplets communs à R1 et R2

On notera :

$$\mathbf{R3 = R1 \cap R2}$$

R1			R2	
A	B	$\cap$	A	B
0	1		0	1
2	3		4	5

$$\mathbf{R3 = R1 \cap R2}$$



R3	
A	B
0	1



## □ DIFFÉRENCE

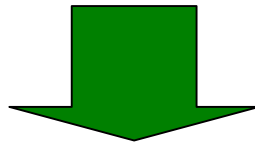
La différence entre deux relations R1 et R2 de même schéma est une relation R3 de schéma identique ayant pour n-uplets les n-uplets de R1 n'appartenant pas à R2

On notera :

$$\mathbf{R3 = R1 - R2}$$

R1		R2	
A	B	A	B
0	1	0	1
2	3	4	5

$$\mathbf{R3 = R1 - R2}$$



R3	
A	B
2	3

## ▣ 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 n-uplets identiques

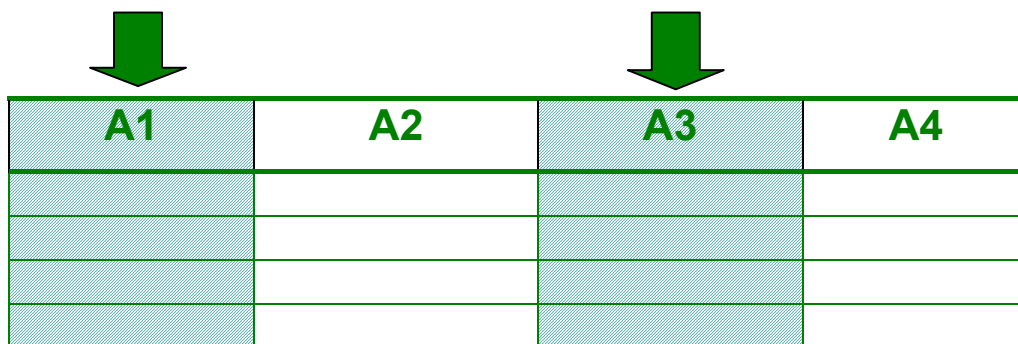
On notera :

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

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 :

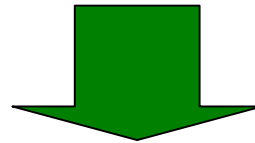
« Quels sont les références et les prix des produits ? »

### 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 :

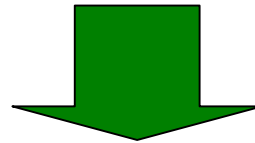
« Quelles sont les marques des produits ? »

**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 :

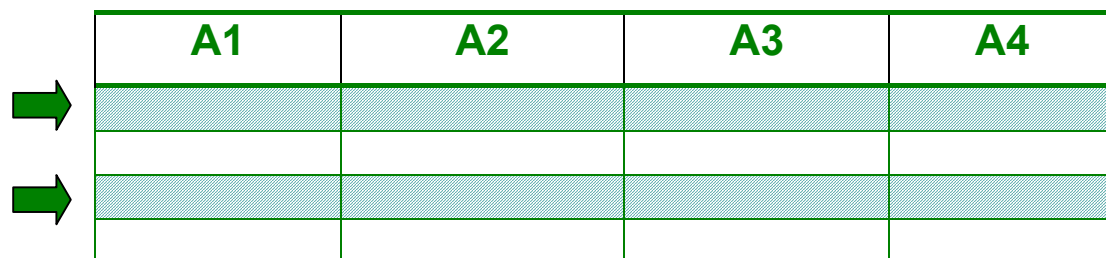
$$R2 = \sigma R1 (\text{condition})$$

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 :

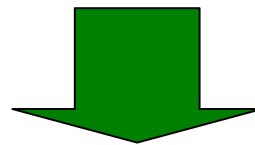
« Quelles sont les produits de marque 'IBM' ? »

#### 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

$\sigma$ 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 n-uplets de R1 avec ceux de R2 et en ne gardant que ceux qui vérifient la condition de liaison

On notera :

$$\mathbf{R3 = R1 \times R2 (condition)}$$

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  $R1 \times R2 (condition)$  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 :  $\text{attribut1} \in 1^{\text{ère}} \text{ relation}$  et  $\text{attribut2} \in 2^{\text{ème}} \text{ relation}$   
:: 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)**

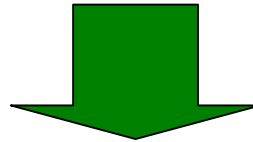
A	B	C
A1	B1	10
A2	B2	10
A3	B3	20
A4	B4	30



**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 »

#### VENTE As V

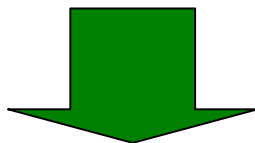
IdCli	IdPro	Date	Qte
X	P	1/1/98	1
Y	Q	2/1/98	1
Z	P	3/1/98	1

×

#### PRODUIT As P

IdPro	Désignation	Prix
P	PS	100
Q	Mac	100

#### VENTE × PRODUIT (V.IdPro=P.IdPro)



Idcli	IdPro	Date	Qte	Désignation	Prix
X	P	1/1/98	1	PS	100
Y	Q	2/1/98	1	Mac	100
Z	P	3/1/98	1	PS	100

- 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 ? »

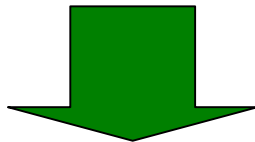
**CLIENT As C1**

IdCli	Nom	Ville
X	Smith	Nice
Y	Blake	Paris
Z	John	Nice

**CLIENT As C2**

IdCli	Nom	Ville
X	Smith	Nice
Y	Blake	Paris
Z	John	Nice

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



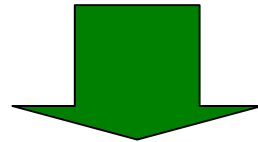
**R1**

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

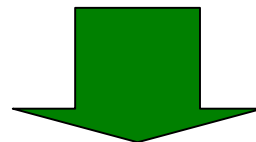
**R2 =  $\sigma_{R1}$  (C2.Nom = 'John')**



**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

$R1 (A1, A2, \dots, An, B1, B2, \dots, Bm)$

$R2 (B1, B2, \dots, Bm)$

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 ? »

**INSCRIPT**

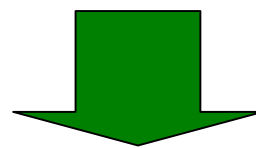
Elève	Sport
toto	judo
tata	danse
toto	foot
toto	danse

÷

**SPORT**

Sport
judo
foot
danse

**RES = INSCRIPT ÷ SPORT**



**RES**

Elève
toto

## II 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 F »

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

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

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

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

## Requête 9 :

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

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

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

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

## 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 »

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

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

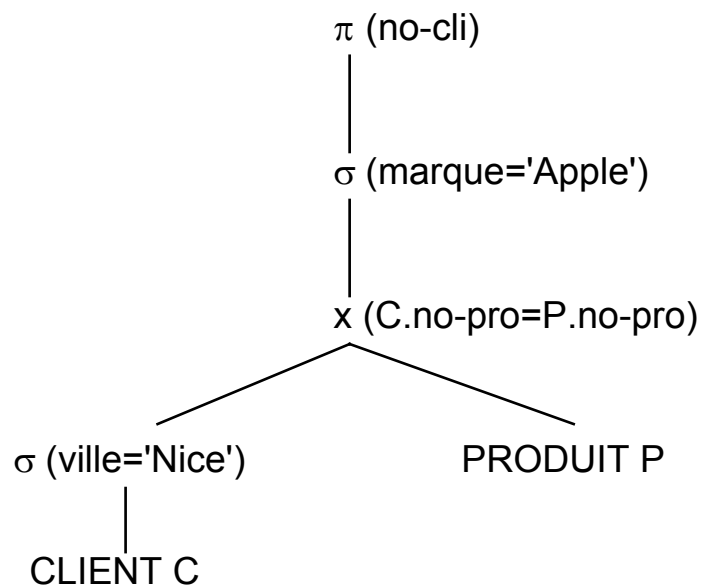
**$R3 = \pi_{\text{VENTE}} (\text{IdCli}, \text{IdPro})$**

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

## □ Arbre algébrique

une question peut être représentée par un arbre

« Quels sont les clients de Nice ayant acheté un produit de marque 'Apple' ? »



## □ Optimisation de requêtes

Plusieurs arbres équivalents peuvent être déduits d'un arbre donné à l'aide de règles de transformation simples, telles que permutation des jointures et restrictions, permutation des projections et des jointures, etc.

Ces transformations sont à la base des techniques  
**d'optimisation de requêtes**

## **Chapitre 5      Les langages de requête, QBE, SQL**

### **I    Les langages associés au modèle relationnel**

### **II   QBE (Query By Example)**

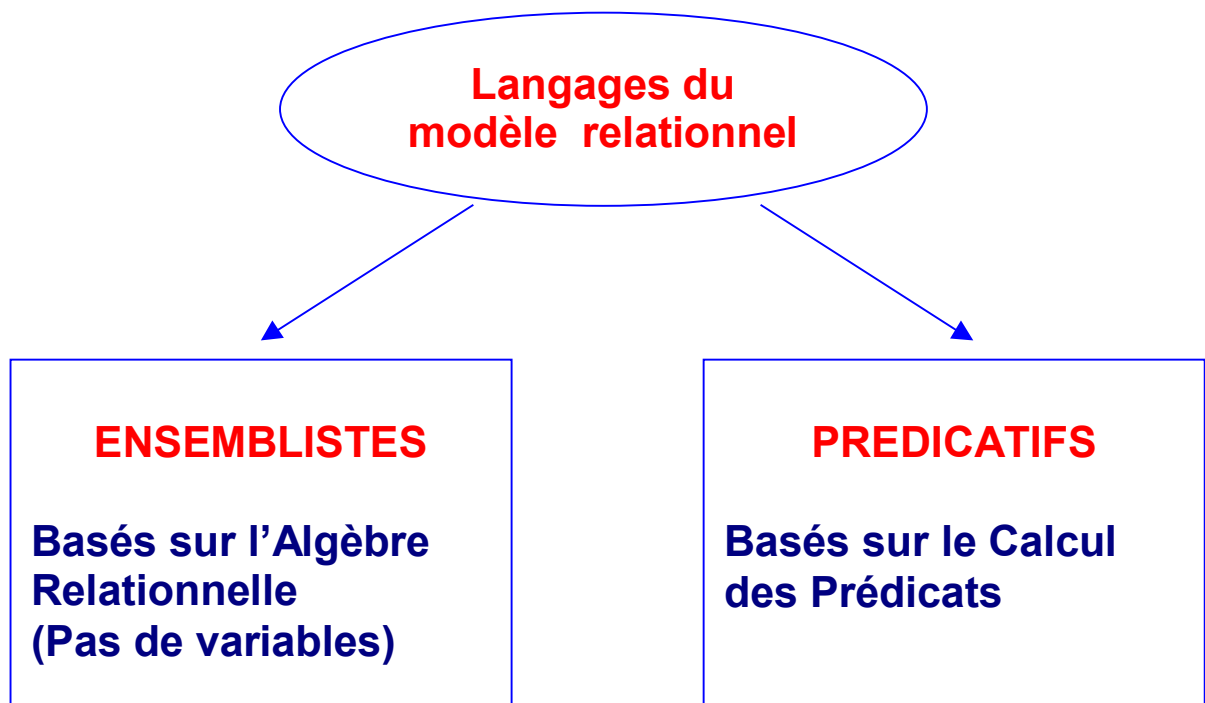
### **III   SQL (Structured Query Language)**

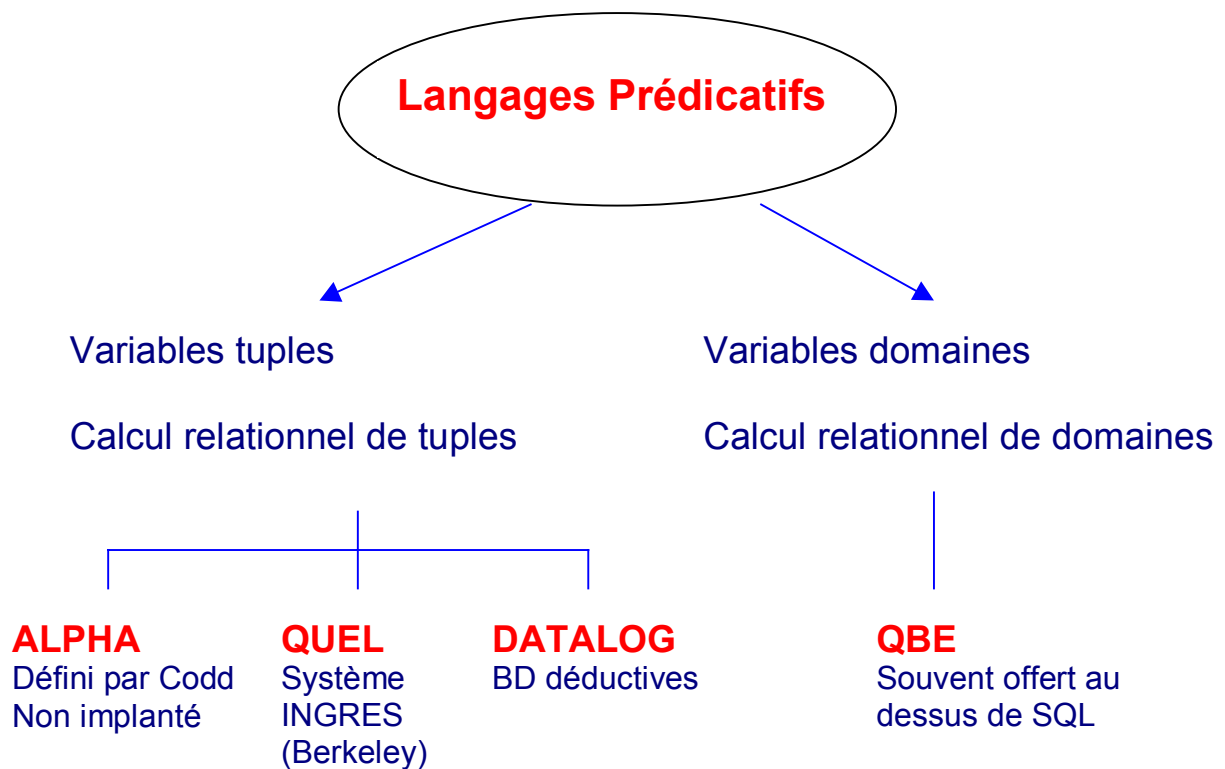
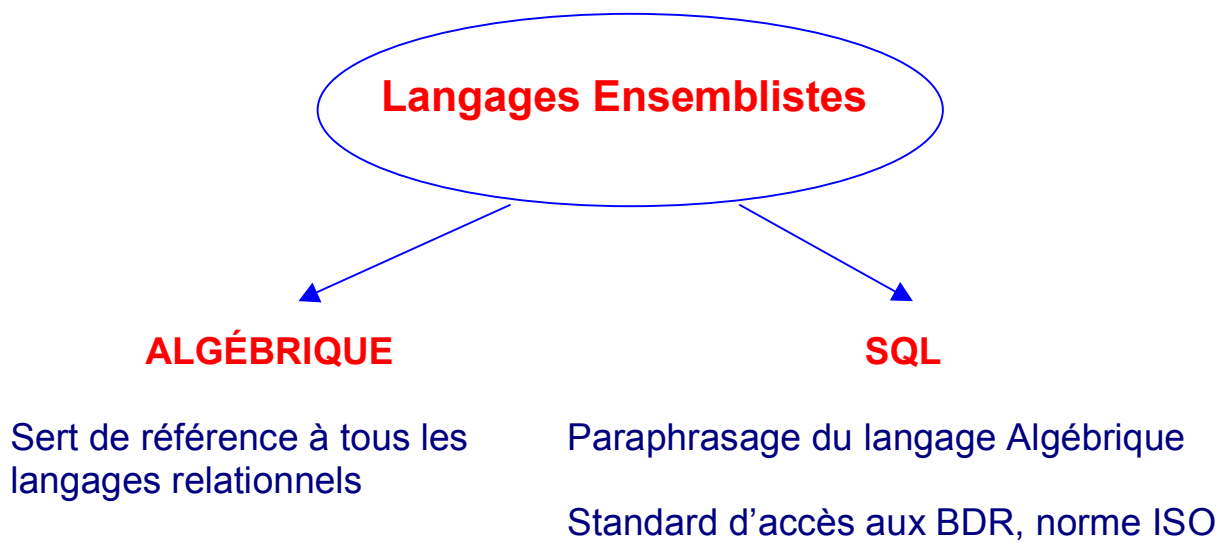
# I Les langages associés au modèle relationnel

- Langages **assertionnels** pour décrire et manipuler les BD relationnelles

ils permettent de spécifier les ensembles de données à sélectionner ou à mettre à jour à partir de propriétés des valeurs (**qualification**), sans dire comment retrouver les données : « *dire QUOI sans dire COMMENT* »

- Deux classes de langages correspondant à la manière de considérer une relation : comme un **ensemble**, ou comme un **prédicat**





- Exemple d'une requête dans les 4 langages ALGÈBRIQUE, SQL, ALPHA, et QBE

Donner les noms des clients qui ont acheté le produit 'p1' ?

## ALGÈBRIQUE

$R1 = CLIENT \times VENTE (CLIENT.IdCli = VENTE.IdCli)$

$R2 = \sigma R1 (IdPro = 'p1')$

$R3 = \pi R2 (Nom)$

## SQL

```
SELECT C.Nom
FROM   client C, vente V
WHERE  C.IdCli = V.IdCli
        AND V.IdPro = 'p1'
```

## ALPHA

```
RANGE OF v IS vente      -- v est une variable tuple
GET client.Nom :          -- qualification
     $\exists v (v.IdCli = client.IdCli \wedge v.IdPro = 'p1')$ 
```

## QBE

CLIENT	IdCli	Nom	Ville	
	<u>x</u>	P.		

VENTE	IdCli	IdPro	Date	Qte	
	<u>x</u>	p1			

x est une variable domaine (*valeur exemple*)

## II QBE (Query By Example)

- QBE a été développé par IBM (Zloof 77)
- L'idée de base de QBE est de formuler une question par un exemple de réponse dans les emplacements appropriés d'une table vide

Ex.: Quels sont les noms des clients de Nice ?

CLIENT	IdCli	nom	ville	
		P. <u>toto</u>	Nice	

P. signifie "print" ; il indique le but de la question

toto est une *valeur exemple*, i.e. un exemple de réponse possible (les valeurs exemples sont soulignées)

Nice (non souligné) est une constante

- Les variables domaines sont désignées par des **valeurs exemples soulignées**  
elles servent pour établir des liens entre des lignes  
si aucun lien n'est nécessaire, comme dans l'exemple ci-dessus, on peut omettre la valeur exemple (P.toto peut être réduit à P.)
- Une variable non imprimée non précédée de P. est implicitement quantifiée par "il existe"
- Une variable peut être quantifiée par "quel-que-soit" en tapant ALL. devant son nom
- La disjonction (ou) est exprimée en utilisant 2 lignes (2 exemples)

# 1 Opérations de recherche

## □ Recherche simple

Q1 Donner les nos des produits vendus

VENTE		IdCli		IdPro		date		qte	
				P. <u>p</u>					

QBE élimine automatiquement les doublons  
pour obtenir les doublons il faut spécifier ALL.

VENTE		IdCli		IdPro		date		qte	
				P.ALL. <u>p</u>					

Q2 Donner tous les renseignements sur tous les clients

CLIENT		IdCli		nom		ville	
		P. <u>x</u>		P. <u>toto</u>		P. <u>rome</u>	

on peut abrégé en plaçant P. en tête

CLIENT		IdCli		nom		ville	
P.							



## □ Recherche qualifiée

Q3 Donner les nos des produits de marque Apple et coûtant moins de 8000 F.

PRODUIT	IdPro	nom	marque	prix
	P. <u>p</u>		Apple	<8000

Q4 Donner les noms des clients de Nice ou Rome

CLIENT	IdCli	nom	ville
	P. <u>x</u>		Nice
	P. <u>y</u>		Rome

- Le ou est exprimé en utilisant 2 lignes (2 exemples)

Q5 Donner les nos des clients qui ont acheté à la fois le produit p1 et le produit p2

VENTE	IdCli	IdPro	date	qte
	P. <u>x</u>	p1		
	<u>x</u>	p2		

## □ Recherche avec tri

Q6 Donner les noms des clients triés par villes et noms

CLIENT	IdCli	nom	ville
		P.AO(2). <u>toto</u>	P.AO(1). <u>nice</u>

il faut spécifier AO. ou DO. pour obtenir des résultats triés par ordre croissant ou décroissant ; on peut préciser le champ majeur lorsque le tri s'effectue sur plusieurs champs

## □ Recherche utilisant un lien

Q7 Donner les noms des clients qui ont acheté le produit p1

CLIENT	IdCli	nom	ville	
	<u>x</u>	P. <u>toto</u>		
VENTE	IdCli	IdPro	date	qte
	<u>x</u>	p1		

- la valeur exemple x est utilisée comme un lien entre CLIENT et VENTE

## □ Recherche utilisant plusieurs liens

Q8 Donner les noms des clients qui ont acheté au moins un produit de marque 'Apple'

CLIENT	IdCli	nom	ville	
	<u>x</u>	P. <u>toto</u>		
PRODUIT	IdPro	nom	marque	prix
	<u>p</u>		Apple	
VENTE	IdCli	IdPro	date	qte
	<u>x</u>	<u>p</u>		

## □ Recherche utilisant la négation

Q9 Donner les noms des clients qui n'ont pas acheté le produit p1

CLIENT		IdCli		nom		ville	
		<u>x</u>		P. <u>toto</u>			

VENTE		IdCli		IdPro		date		qte	
¬		<u>x</u>		p1					

- Notez l'opérateur négation (¬) portant sur la ligne de VENTE

## □ Recherche utilisant un lien dans une seule table

Q10 Donner les nos des clients qui ont acheté au moins un produit acheté par le client c1

VENTE		IdCli		IdPro		date		qte	
		P. <u>x</u>		<u>p</u>					
		c1		<u>p</u>					

## □ Recherche utilisant une boîte condition

Q11 Donner les nos des produits achetés par plus qu'un seul client

VENTE		IdCli		IdPro		date		qte	
		<u>x</u>		P. <u>p</u>					
		<u>y</u>		<u>p</u>					
		CONDITION BOX							
		<u>x ≠ y</u>							

- Lorsque la condition est importante ou si l'on veut simplifier l'écriture d'une requête on peut utiliser une *boîte condition*

## 2 Fonctions de calcul

- QBE fournit un ensemble de fonctions de calcul prédéfinies :

CNT . ALL	CNT . UNQ . ALL
SUM . ALL	SUM . UNQ . ALL
AVG . ALL	AVG . UNQ . ALL
MAX . ALL	
MIN . ALL	

- ALL. est toujours spécifié
- L'option UNQ. signifie "éliminer les doublons avant d'appliquer la fonction"
- Les valeurs nulles sont éliminées, sauf pour CNT.

### □ Recherche simple utilisant une fonction

Q12 Donner le nombre total de clients

CLIENT	IdCli	nom	ville	
	P.CNT.ALL.x			

Q13 Donner le nombre total de clients ayant acheté des produits

VENTE	IdCli	IdPro	date	qte	
	P.CNT.UNQ.ALL.x				

## □ Recherche qualifiée utilisant une fonction

Q14 Donner le nombre de ventes du produit p1

VENTE	IdCli	IdPro	date	qte	
	P.CNT.ALL. <u>x</u>	p1			

Q15 Donner la qte totale vendue du produit p1

VENTE	IdCli	IdPro	date	qte	
		p1		P.SUM.ALL. <u>qte</u>	

## □ Fonction dans la boîte condition

Q16 Donner les nos des produits moins chers que tous ceux de marque Apple

PRODUIT	IdPro	nom	marque	prix	
	P. <u>p</u>			<u>val</u>	
			Apple	<u>prix</u>	

	CONDITION BOX	
	<u>val</u> < MIN.ALL. <u>prix</u>	

## □ Recherche avec regroupement

Q17 Pour chaque produit vendu, donner le no de produit et la quantité totale vendue de ce produit

VENTE	IdCli	IdPro	date	qte	
		P.G. <u>p</u>		P.SUM.ALL. <u>qte</u>	

G. est l'opérateur de regroupement

### 3 Opérations de mise à jour

- Les opérateurs de mise à jour sont :
  - **UPDATE** (modification)
  - **INSERT** (insertion)
  - **DELETE** (suppression)

Le nom des opérateurs apparaît sous le nom de la relation abrégé par U. , I. ou D.

#### □ Modification d'un seul enregistrement

- L'enregistrement à modifier est identifié par la valeur de sa clé primaire
- Les valeurs de clé primaire ne peuvent pas être modifiées

Q18 Changer la ville du client c1 par Nice

CLIENT	IdCli	nom	ville
U.	c1		Nice

Q19 Augmenter de 100F le prix du produit p1

PRODUIT	IdPro	nom	marque	prix
	p1			<u>val</u>
U.	p1			<u>val</u> + 100

## □ Modification de plusieurs enregistrements

- Les enregistrements à modifier sont spécifiés par une valeur exemple (non une constante) de clé primaire

Q20 Doubler le prix de tous les produits Apple

PRODUIT	IdPro	nom	marque	prix
	<u>p</u>		Apple	<u>val</u>
U.	<u>p</u>			2 * <u>val</u>

Q21 Mettre à 0 toutes les qtés achetées par les clients de Nice

CLIENT	IdCli	nom	ville
	<u>x</u>		Nice

VENTE	IdCli	IdPro	date	qte
U.	<u>x</u>			0

## □ Insertion d'un nouvel enregistrement

Q22 Ajouter le client (c20, 'Duduche', 'Nice') dans la table CLIENT

CLIENT	IdCli	nom	ville
I.	c20	Duduche	Nice

- Le nouvel enregistrement doit avoir une valeur de clé primaire non nulle, et distincte de toutes les valeurs de clés primaires existantes dans la table

## □ Insertion de plusieurs enregistrements

Q23 La table TEMP a une seule colonne IdPro. Introduire dans TEMP les nos de tous les produits achetés par le client c1

TEMP	I.	IdPro	
		<u>p</u>	

VENTE	IdCli	IdPro	date	qte
	c1	<u>p</u>		

## □ Suppression d'un seul enregistrement

Q24 Supprimer le client c1

CLIENT	D.	IdCli	nom	ville
		c1		

- La suppression n'est pas possible si le client figure dans la table VENTE (*intégrité référentielle*)

## □ Suppression de plusieurs enregistrements

Q25 Supprimer tous les clients de Nice

CLIENT	D.	IdCli	nom	ville
				Nice

- A nouveau la suppression n'est possible qu'en respect de l'*intégrité référentielle* avec VENTE



## 4 Dictionnaire de données QBE

- Le **dictionnaire de données** contient la description des relations du schéma relationnel de la BD
- Le DD est représenté sous forme de relations prédéfinies, parmi lesquelles :

**CATALOG** (TNAME, CREATOR, NCOLS, ...)

informations sur toutes les tables de la BD

**COLUMNS** (TNAME, CNAME, TYPE, LENGTH, ...)

informations sur les colonnes de toutes les tables de la BD

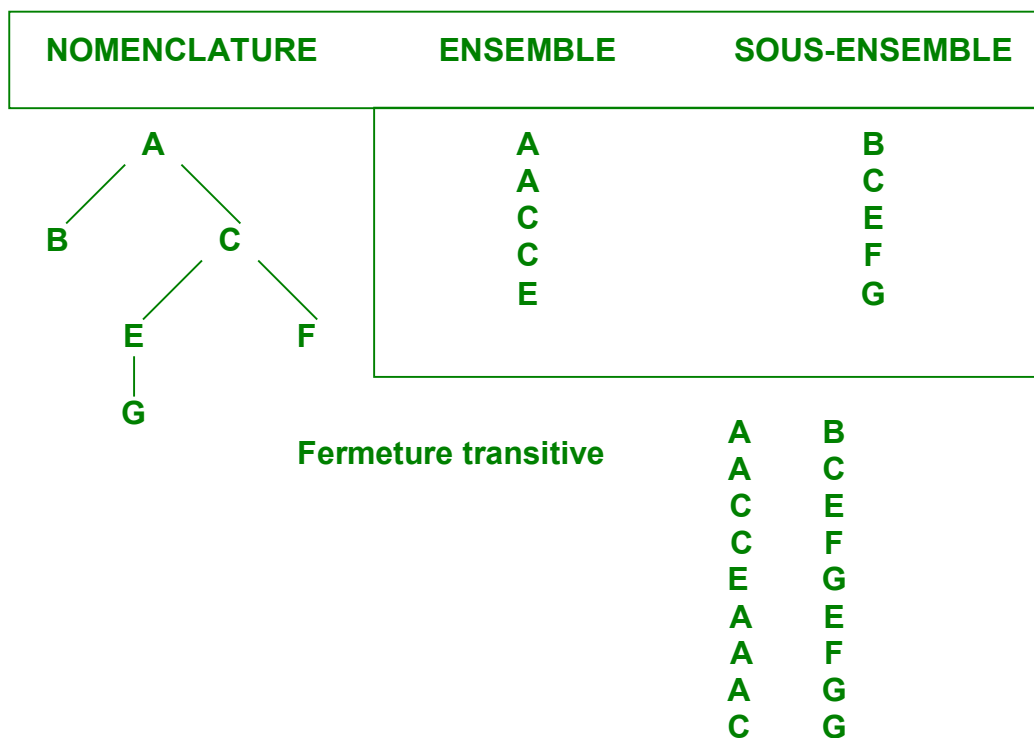
- Le DD peut être interrogé et mis à jour avec le langage de manipulation de données

## 5 Fermeture transitive

- La **fermeture transitive** permet d'enrichir une relation à 2 attributs de même domaine, avec tous les couples qui se déduisent par transitivité

Si on a (a, b) et (b, c) alors on ajoute (a, c)

Ex.



- La fermeture transitive ne peut pas être exprimée par une expression constante de l'algèbre relationnelle  
elle peut être effectuée par une série de jointure/projection/union, mais le nbre d'opérations à effectuer dépend du contenu de la relation

- QBE permet d'exprimer des questions de type fermeture transitive

Q26 Donner les sous-ensembles au 2ième niveau de l'ensemble A

NOMENCLATURE	ensemble	sous-ens	
	A	<u>u</u>	
	<u>u</u>	P. <u>v</u>	

Pour rechercher les sous-ensembles de 3ième niveau il faut ajouter une ligne :

NOMENCLATURE	ensemble	sous-ens	
	A	<u>u</u>	
	<u>u</u>	<u>v</u>	
	<u>v</u>	P. <u>w</u>	

Pour rechercher les sous-ensembles de niveau n, il faut n lignes ; QBE fournit un raccourci en permettant d'indiquer le nombre de niveau :

NOMENCLATURE	ensemble	sous-ens	
	A	P. <u>u</u> (3L)	

(3L) signifie "third level"

QBE permet aussi d'obtenir les sous-ensembles de dernier niveau à l'aide du mot clé LAST

### III SQL (Structured Query Language)

- Introduit par IBM, évolution du langage SEQUEL, commercialisé tout d'abord par ORACLE
- SQL est devenu le langage standard pour décrire et manipuler les BDR
- Les commandes SQL :
  - De définition des données :  
**CREATE**  
**DROP**  
**ALTER**
  - De manipulation des données :  
**SELECT**  
**INSERT**  
**UPDATE**  
**DELETE**
  - De contrôle des données :
    - ➔ Contrôle des accès concurrents  
**COMMIT**  
**ROLLBACK**
    - ➔ Contrôle des droits d'accès  
**GRANT**  
**REVOKE**

- SQL peut être utilisé de 2 manières :

- en mode interactif

→ **pour apprendre le langage**

SQL est un langage pour les développeurs  
n'est pas destiné à un utilisateur final

Les requêtes sont envoyées à partir d'un terminal interactif  
auquel les résultats sont retournés

Ex. :

```
SELECT C.ville
FROM   client C
WHERE  C.IdCli = 'c1'
```

- en mode intégré dans un L3G hôte  
(COBOL, ADA, C, FORTRAN ...)

→ **pour développer des applications**

Les constantes dans les requêtes SQL peuvent être  
remplacées par des **variables du programme hôte** ; les  
résultats doivent être transmis dans des variables

Ex. : SQL dans C

```
EXEC SQL    SELECT C.ville INTO :laVille
            FROM   client C
            WHERE  C.IdCli = :unIdCli ;
```

les variables du programme sont précédées par (:)

- La notion de curseur permet d'exploiter les résultats  
d'une requête ligne à ligne
- Un programme intégrant SQL doit être précompilé par  
un précompilateur SQL

# 1 Importance du langage SQL

- Standard d'accès aux serveurs de données relationnels, norme **ISO**
- SQL est le langage commun de nombreux systèmes commercialisés
- SQL est l'interface logiciel/logiciel entre les applications et les BDR

**Applications**

---

**SQL**

---

**ORACLE**

**DB2**

**INGRES**

**SYBASE**

**INFORMIX**

- Plusieurs niveaux de normalisation
  - **SQL1** : norme de base
  - **SQL2** : extension de SQL1
    - meilleur support des règles du relationnel
    - types de données plus variés
  - **SQL3** : intégration du modèle objet
- Quels sont les avantages de la normalisation ?
  - ➔ Réduction des coûts de formation
  - ➔ Portabilité des applications
  - ➔ Pérennité des applications
  - ➔ Communication facilitée entre systèmes

## 2 Définition des données

### ▣ CRÉATION DE TABLES

La commande **CREATE TABLE** crée la définition d'une table

**Syntaxe :**

```
CREATE TABLE table
(
    -- définition des colonnes
    colonne    type    [ NOT NULL [UNIQUE] ]
                                   [ DEFAULT valeur ]
                                   [ PRIMARY KEY ]
                                   [ REFERENCES table ]
                                   [ CHECK condition ]      ,
    ...
    -- contraintes de table
    [ PRIMARY KEY (liste de colonnes) ],
    [ UNIQUE (liste de colonnes) ]      ,
    ...
    [ FOREIGN KEY (liste de colonnes) REFERENCES
table
    [ ON DELETE {RESTRICT | CASCADE | SET NULL} ]
    [ ON UPDATE {RESTRICT | CASCADE | SET NULL} ] ,
    ...
    [ CHECK condition ]                ,
    ...
)
```



- **Principaux types de données**

**CHAR(n)**

**SMALLINT**

**INTEGER**

**DECIMAL(n,m)**

**DATE**

- **Contraintes d'intégrité**

**NOT NULL**                      valeur null impossible

**UNIQUE**                        unicité d'un attribut

**PRIMARY KEY**                clé primaire

**FOREIGN KEY**                clé étrangère

**CHECK**                        plage ou liste de valeurs

Une contrainte qui ne fait référence qu'à une seule colonne de la table peut faire partie intégrante de la définition de colonne

- Toute opération de mise à jour violant une des contraintes spécifiées sera rejetée

## Le système garantit l'intégrité des données

- SQL2 permet de spécifier les actions à entreprendre pour le maintien de l'intégrité référentielle, lors d'une suppression ou d'une modification d'un tuple référencé

**CASCADE**      cascader les suppressions ou modifications

par ex. si on supprime un produit dans la table PRODUIT, toutes les ventes correspondantes seront supprimées dans la table VENTE

**SET NULL**      rendre nul les attributs référençant

par ex. si on modifie la référence d'un produit dans la table PRODUIT, toutes les références correspondantes seront modifiées dans la table VENTE

**RESTRICT**      rejet de la mise à jour  
c'est l'option par défaut

- Exemple

```
CREATE TABLE client
(
    IdCli    CHAR(4)    PRIMARY KEY
    ,
    nom      CHAR(20)
    ,
    ville    CHAR(30)
    ,
    CHECK (ville IN ('Nice', 'Paris', 'Rome'))
)
```

```
CREATE TABLE produit
(
    IdPro    CHAR(6)    PRIMARY KEY
    ,
    nom      CHAR(30)    NOT NULL UNIQUE
    ,
    marque   CHAR(30)
    ,
    prix     DEC(6,2)
    ,
    qstock   SMALLINT
    ,
    CHECK (qstock BETWEEN 0 AND 100)
    ,
    -- contrainte de table
    CHECK (marque <> 'IBM' OR qstock < 10)
)
```

```
CREATE TABLE vente
(
    IdCli    CHAR(4)    NOT NULL
    ,
    REFERENCES client
    ,
    IdPro    CHAR(6)    NOT NULL
    ,
    date     DATE       NOT NULL
    ,
    qte      SMALLINT
    ,
    CHECK (qte BETWEEN 1 AND 10)
    ,
    -- contrainte de table
    PRIMARY KEY (IdCli, IdPro, date)
    ,
    FOREIGN KEY (IdPro) REFERENCES produit
    ON DELETE CASCADE ON UPDATE CASCADE
)
```

## ▣ CRÉATION D'INDEX

La commande **CREATE INDEX** permet de créer des index multi-colonne

Syntaxe :

```
CREATE [UNIQUE] INDEX index  
ON table (colonne [ASC|DESC], ...)
```

- L'option **UNIQUE** permet d'assurer l'unicité d'une clé

Ex.: **CREATE UNIQUE INDEX index1 ON client(Nom)**

- Les index permettent d'accélérer les recherches
- Le système détermine sa stratégie d'accès en fonction des index existants
- Les index sont automatiquement mis à jour
- Il est indispensable de créer les index appropriés pour accélérer le traitement des requêtes
- Il ne faut cependant pas créer des index sur n'importe quel colonne ou groupe de colonnes, car les mises à jour seraient ralenties inutilement par la maintenance de ces index
- Un index est supprimé par la commande **DROP INDEX**

## ▣ MODIFICATION DU SCHÉMA

La modification du schéma n'est pas prévue dans SQL1 ; cependant la plupart des systèmes permettent la suppression ou la modification d'une table à l'aide des commandes :

**DROP TABLE**

**ALTER TABLE**

Ex.:

```
ALTER TABLE client  
    ADD COLUMN teleph CHAR(16)
```

## ▣ DICTIONNAIRE DE DONNÉES

Le **dictionnaire de données** contient la description de tous les objets (relations, index, ...) de la BD

- Le DD est décrit sous forme de **tables systèmes**

- Par exemple, on peut citer dans DB2 :

**SYSTABLES (NAME, CREATOR, COLCOUNT, ...)**  
description des tables

**SYSCOLUMNS (NAME, TBNAME, COLTYPE, ...)**  
description des colonnes

- Le DD peut être consulté de la même manière que les tables de base avec le langage d'interrogation  
il faut toutefois connaître les noms et les schémas des tables systèmes

### 3 Manipulation des données

**SELECT**, **INSERT**, **UPDATE** et **DELETE** sont les 4 commandes de manipulation des données en SQL

Ex. :

#### Recherche **SELECT**

```
SELECT    P.prix  
FROM      produit P  
WHERE     P.IdPro = 'p1'
```

#### Ajout **INSERT**

```
INSERT  
INTO      client (IdCli, nom, ville)  
VALUES    ('c100', 'Duduche', 'Nice')
```

#### Mise à jour **UPDATE**

```
UPDATE    produit P  
SET       P.prix = P.prix * 1.20  
WHERE     P.IdPro = 'p2'
```

#### Suppression **DELETE**

```
DELETE  
FROM      produit P  
WHERE     P.IdPro = 'p4'
```

## ▣ LA COMMANDE SELECT

La commande **SELECT** permet de rechercher des données à partir de plusieurs tables ; le résultat est présenté sous forme d'une **table réponse**

### • Expression des projections

Q1 Donner les noms, marques et prix des produits

```
SELECT      P.nom, P.marque, P.prix  
FROM        produit P
```

### Synonyme de nom de table (ou **alias**)

- On peut introduire dans la clause FROM un synonyme (*alias*) à un nom de table en le plaçant immédiatement après le nom de la table
- Les noms de table ou les synonymes peuvent être utilisés pour préfixer les noms de colonnes dans le SELECT
- Les préfixes ne sont obligatoires que dans des cas particuliers (par ex. pour une auto-jointure) ; leur emploi est cependant conseillé pour la clarté
- Un alias est utilisé par SQL comme une variable de parcours de table (dite *variable de corrélation*) désignant à tout instant une ligne de la table



Q2 Donner les différentes marques de produit

```
SELECT      DISTINCT P.marque  
FROM        produit P
```

- Contrairement à l'algèbre relationnelle, SQL n'élimine pas les doublons
- Pour éliminer les doublons il faut spécifier **DISTINCT**

Q3 Donner les références des produits et leurs prix majorés de 20%

```
SELECT      P.IdPro, P.prix * 1.20  
FROM        produit P
```

- Il est possible d'effectuer des opérations arithmétiques (+, -, \*, /) sur les colonnes extraites

Q4 Donner tous les renseignements sur les clients

```
SELECT      *  
FROM        client
```

- Une étoile (\*) permet de lister tous les attributs

## • Expression des restrictions

Q5 Donner les noms des produits de marque IBM

```
SELECT      P.nom  
FROM        produit P  
WHERE       P.marque = 'IBM'
```

- La condition de recherche (qualification) est spécifiée après la clause **WHERE** par un prédicat
- Un prédicat simple peut-être :
  - un prédicat d'égalité ou d'inégalité (=, <>, <, >, <=, >=)
  - un prédicat **LIKE**
  - un prédicat **BETWEEN**
  - un prédicat **IN**
  - un test de valeur **NULL**
  - un prédicat **EXISTS**
  - un prédicat **ALL** ou **ANY**
- Un prédicat composé est construit à l'aide des connecteurs **AND**, **OR** et **NOT**

## ▪ Exemples

Q6 Lister les clients dont le nom comporte la lettre A en 2<sup>ième</sup> position

```
SELECT      *  
FROM        client C  
WHERE       C.nom LIKE '_A%'
```

Le prédicat **LIKE** compare une chaîne avec un modèle  
(  ) remplace n'importe quel caractère  
(%) remplace n'importe quelle suite de caractères

Q7 Lister les produits dont le prix est compris entre 5000F et 12000F

```
SELECT      *  
FROM        produit P  
WHERE       P.prix BETWEEN 5000 AND 12000
```

Le prédicat **BETWEEN** teste l'appartenance à un intervalle

Q8 Lister les produits de marque IBM, Apple ou Dec

```
SELECT      *  
FROM        produit P  
WHERE       P.marque IN ('IBM', 'Apple', 'Dec')
```

Le prédicat **IN** teste l'appartenance à une liste de valeurs

Q9 Lister les produits dont le prix est inconnu

```
SELECT      *  
FROM        produit P  
WHERE       P.prix IS NULL
```

La valeur **NULL** signifie qu'une donnée est inconnue

Q10 Lister les produits de marque IBM dont le prix est inférieur à 12000F

```
SELECT      *  
FROM        produit P  
WHERE       P.marque = 'IBM' AND P.prix < 12000
```

Le connecteur **AND** relie les 2 prédicats de comparaison

## • USER

Le mot réservé **USER** désigne l'usager courant

## • Valeurs nulles

La valeur **NULL** est une valeur particulière signifiant qu'une donnée est manquante, sa valeur est inconnue

- Dans une expression arithmétique, si l'un des termes est null, alors l'expression entière prend la valeur NULL
- Un prédicat de comparaison (=, <>, <, <=, >, >=) prend la valeur logique "inconnu" si l'un des termes de la comparaison est NULL

AND	F	V	?
F	F	F	F
V	F	V	?
?	F	?	?

OR	F	V	?
F	F	V	?
V	V	V	V
?	?	V	?

NOT	
F	V
V	F
?	?

V = vrai, F = faux, ? = inconnu

- **Tri du résultat d'un SELECT**

La clause **ORDER BY** permet de spécifier les colonnes définissant les critères de tri

- Le tri se fera d'abord selon la première colonne spécifiée, puis selon la deuxième colonne etc...
- Exemple

Q11 Lister les produits en les triant par marques et à l'intérieur d'une marque par prix décroissants

```
SELECT      *  
FROM        produit P  
ORDER BY    P.marque, P.prix DESC
```

- L'ordre de tri est précisé par **ASC** (croissant) ou **DESC** (décroissant) ; par défaut ASC

## • Expression des jointures

Le produit cartésien s'exprime simplement en incluant plusieurs tables après la clause **FROM**

- La condition de jointure est exprimée après WHERE
- Exemples :

Q12 Donner les références et les noms des produits vendus

```
SELECT      P.IdPro, P.nom
FROM        produit P , vente V
WHERE       P.IdPro = V.IdPro
```

Q13 Donner les noms des clients qui ont acheté le produit de nom 'PS1'

```
SELECT      C.nom
FROM        client C , produit P, vente V
WHERE       V.IdCli = C.IdCli
AND         V.IdPro = P.IdPro
AND         P.nom = 'PS1'
```

- **Auto-jointure**

Q14 Donner les noms des clients de la même ville que John

```
SELECT      C2.nom
FROM        client C1 , client C2
WHERE       C1.ville = C2.ville
           AND  C1.nom = 'John'
           AND  C2.nom <> 'John'
```

- Cet exemple utilise, pour le couplage des villes, la jointure de la table Client avec elle-même (*auto-jointure*)
- Pour pouvoir distinguer les références ville dans les 2 copies, il faut introduire 2 alias différents C1 et C2 de la table client



## • Jointures externes

La **jointure externe** permet de retenir lors d'une jointure les lignes d'une table qui n'ont pas de correspondant dans l'autre table, avec des valeurs nulles associées

On distingue **jointure externe gauche**, **droite** et **complète** selon que l'on retient les lignes sans correspondant des 2 tables ou seulement d'une

SQL2 offre la possibilité de spécifier les jointures externes au niveau de la clause FROM selon la syntaxe suivante :

```
FROM table1 [NATURAL] [{LEFT|RIGHT}] JOIN table2  
[ON ( liste de colonnes = liste de colonnes) ]
```

**NATURAL** signifie jointure naturelle, c.a.d l'égalité des attributs de même nom

Q15 Lister tous les clients avec le cas échéant leurs achats

```
SELECT      C.IdCli, C.nom, C.ville  
            V.IdPro, V.date, V.qte  
FROM        client C NATURAL LEFT JOIN vente V
```

## • Sous-requêtes

SQL permet l'imbrication de **sous-requêtes** au niveau de la clause WHERE

d'où le terme "*structuré*" dans Structured Query Language

- Les sous-requêtes sont utilisées :
  - dans des prédicats de comparaison (=, <>, <, <=, >, >=)
  - dans des prédicats **IN**
  - dans des prédicats **EXISTS**
  - dans des prédicats **ALL** ou **ANY**
- Une sous-requête dans un prédicat de comparaison doit se réduire à une seule valeur ("**singleton select**" )
- Une sous-requête dans un prédicat IN, ALL ou ANY doit représenter une table à colonne unique
- L'utilisation de constructions du type "IN sous-requête" permet d'exprimer des jointures de manière procédurale ... *ce qui est déconseillé !!*

## ▪ Exemple

Q16 Donner les noms des clients qui ont acheté le produit 'p1'

- Avec sous-requête

```
SELECT      C.nom
FROM        client C
WHERE       IdCli IN
            (
              SELECT V.IdCli
              FROM   vente V
              WHERE  P.IdPro = 'p1'
            )
```

- Avec jointure

```
SELECT      C.nom
FROM        client C , vente V
WHERE       C.IdCli = V.IdCli
AND V.IdPro = 'p1'
```

♥ De préférence, utiliser la jointure

## • Requêtes quantifiées

### • Le prédicat EXISTS

Il permet de tester si le résultat d'une sous-requête est vide ou non

Q17 Donner les noms des produits qui n'ont pas été acheté

```
SELECT      C.nom
FROM        produit P
WHERE       NOT EXISTS
            ( SELECT *
              FROM    vente V
              WHERE    V.IdPro = P.IdPro )
```

Il permet de répondre à des questions quantifiées par  
"pour tout..." :  $\forall x \mid P(x) \Leftrightarrow \neg ( \exists x \mid \neg P(x) )$

Q18 Donner les noms des produits qui ont été achetés par **tous**  
les clients de Nice

```
SELECT      P.nom
FROM        produit P
WHERE       NOT EXISTS
            (
              SELECT *
              FROM    client C
              WHERE    C.ville = 'Nice'
              AND NOT EXISTS
                    (
                      SELECT *
                      FROM    vente V
                      WHERE    C.IdCli = V.IdCli
                           AND V.IdPro = P.IdPro
                    )
            )
```

- **Le prédicat ALL ou ANY**

Ils permettent de tester si un prédicat de comparaison est vrai pour tous (**ALL**) ou au moins un (**ANY**) des résultats d'une sous-requête

Q19 Donner les nos des clients ayant acheté un produit en quantité supérieure à chacune des quantités de produits achetées par le client 'c1'

```
SELECT      V.IdCli
FROM        vente V
WHERE       V.qte >= ALL
            (
              SELECT W.qte
              FROM   vente W
              WHERE  W.IdCli = 'c1'
            )
```

Q20 Donner les nos des clients ayant acheté un produit en quantité supérieure à au moins l'une des quantités de produits achetées par le client 'c1'

```
SELECT      V.IdCli
FROM        vente V
WHERE       V.qte >= ANY
            (
              SELECT W.qte
              FROM   vente W
              WHERE  W.IdCli = 'c1'
            )
```

Les prédicats ALL et ANY sont redondants, ils peuvent s'exprimer avec EXISTS

$x \Delta ANY$   
( SELECT y  
FROM t  
WHERE p )  
 $\Leftrightarrow$   
EXISTS  
( SELECT \*  
FROM t  
WHERE p AND  $x \Delta t.y$  )

$x \Delta ALL$   
( SELECT y  
FROM t  
WHERE p )  
 $\Leftrightarrow$   
NOT EXISTS  
( SELECT \*  
FROM t  
WHERE p AND NOT (  $x \Delta t.y$  ) )

Où  $\Delta$  est un prédicat de comparaison (=, <>, <, <=, >, >=)

## • Expression des unions

SQL1 permet d'exprimer l'opération d'union en connectant des SELECT par des **UNION**

Q21 Donner les nos des produits de marque IBM ou ceux achetés par le client no 'c1'

```
SELECT      P.IdPro
FROM        produit P
WHERE       P.marque = 'IBM'
```

UNION

```
SELECT      P.IdPro
FROM        vente V
WHERE       V.IdCli = 'c1'
```

- L'union élimine les doublons, pour obtenir les doublons il faut spécifier **ALL** après UNION
- UNION est une opération binaire, on peut écrire :  
(x UNION y) UNION z ou x UNION (y UNION z)
- Les parenthèses sont nécessaires dans certains cas, par ex. :

(x UNION ALL y) UNION z

n'est pas équivalent à

x UNION ALL (y UNION z)

## • Fonctions de calculs

SQL fournit des fonctions de calcul opérant sur l'ensemble des valeurs d'une colonne de table

<i>COUNT</i>	nombre de valeurs
<i>SUM</i>	somme des valeurs
<i>AVG</i>	moyenne des valeurs
<i>MAX</i>	plus grande valeur
<i>MIN</i>	plus petite valeur

Q22 Donner le nombre total de clients

SELECT	COUNT ( IdCli )
FROM	client

Q23 Donner le nombre total de clients ayant acheté des produits

SELECT	COUNT ( DISTINCT IdCli )
FROM	vente

- On peut faire précéder l'argument du mot clé **DISTINCT** pour indiquer que les valeurs redondantes doivent être éliminées avant application de la fonction



- La fonction spéciale **COUNT (\*)** compte toutes les lignes dans une table
- Les valeurs nulles ne sont pas prises en compte, sauf pour COUNT(\*)
- Si l'argument est un ensemble vide, COUNT renvoie la valeur 0, les autres fonctions renvoyant la valeur NULL
- Exemples :

Q24 Donner le nombre total de 'PS1' vendus

```
SELECT      SUM ( V.qte )
FROM        vente V , produit P
WHERE       P.IdPro = V.IdPro
            AND P.nom = 'PS1'
```

Q25 Donner les noms des produits moins chers que la moyenne des prix de tous les produits

```
SELECT      P1.nom
FROM        produit P1
WHERE       P1.prix <
            (
              SELECT  AVG ( P2.prix )
FROM        produit P2
            )
```

Cet exemple montre un "**singleton select**" pour calculer la moyenne des prix

## • La clause GROUP BY

La clause **GROUP BY** permet de partitionner une table en plusieurs groupes

- Toutes les lignes d'un même groupe ont la même valeur pour la liste des attributs de partitionnement spécifiés après GROUP BY
- Les fonctions de calcul opèrent sur chaque groupe de valeurs
- Exemples :

Q26 Donner pour chaque référence de produit la quantité totale vendue

```
SELECT      V.IdPro, SUM ( V.qte )  
FROM        vente V  
GROUP BY    V.IdPro
```

Q27 Donner la quantité totale achetée par chaque client (0 pour ceux qui n'ont rien acheté)

```
SELECT      C.IdCli, SUM ( V.qte )  
FROM        client C NATURAL LEFT JOIN vente V  
GROUP BY    C.IdCli
```

## • La clause **HAVING**

La clause **HAVING** permet de spécifier une condition de restriction des groupes

- Elle sert à éliminer certains groupes, comme WHERE sert à éliminer des lignes
- Exemples

Q28 Donner les noms des marques dont le prix moyen des produits est < 5000F

SELECT	P.marque, AVG ( P.prix )
FROM	produit P
GROUP BY	P.marque
HAVING	AVG ( P.prix ) < 5000

Q29 Donner les références des produits achetés en qte > 10 par plus de 50 clients

SELECT	P.marque, AVG ( P.prix )
FROM	vente V
WHERE	V.qte > 10
GROUP BY	V.IdPro
HAVING	COUNT (*) > 50

## • La forme générale de SELECT

<i>SELECT</i>	<i>[DISTINCT]</i> liste d'attributs, expressions
<i>FROM</i>	liste de tables ou vues
<i>WHERE</i>	qualification
<i>GROUP BY</i>	attributs de partitionnement
<i>HAVING</i>	qualification de groupe
<i>ORDER BY</i>	liste de colonnes <i>[ASC   DESC]</i>

### ▪ Exemple

Q30 Donner les nos, les prix, les marques et la quantité maximum vendue de tous les produits IBM, Apple ou Dec dont la quantité totale vendue est supérieure à 500 et dont les quantités vendues sont > 10

SELECT	P.IdPro, P.prix, P.marque, 'Qte max vendue = ', MAX ( V.qte)
FROM	produit P , vente V
WHERE	P.IdPro = V.IdPro AND P.marque IN ('IBM', 'Apple', 'Dec') AND V.qte > 10
GROUP BY	P.IdPro, P.prix, P.marque
HAVING	SUM ( V;qte ) > 500

Du seul point de vue logique, on peut considérer que le résultat d'un **SELECT** est construit suivant les étapes :

### 1. **FROM**

la clause *FROM* est évaluée de manière à produire une nouvelle table, produit cartésien des tables dont le nom figure après *FROM*

### 2. **WHERE**

le résultat de l'étape 1 est réduit par élimination de toutes les lignes qui ne satisfont pas à la clause *WHERE*

### 3. **GROUP BY**

le résultat de l'étape 2 est partitionné selon les valeurs des colonnes dont le nom figure dans la clause *GROUP BY*

dans l'exemple ci-dessus, les colonnes sont P.IdPro, P.prix et P.marque ; en théorie il suffirait de prendre uniquement P.IdPro comme colonne définissant les groupes (puisque le prix et la marque sont déterminés par le no de produit)

*SQL oblige de faire apparaître dans la clause GROUP BY toutes les colonnes qui sont mentionnées dans la clause SELECT*

### 4. **HAVING**

les groupes ne satisfaisant pas la condition *HAVING* sont éliminés du résultat de l'étape 3

### 5. **SELECT**

chacun des groupes génère une seule ligne du résultat

## ▣ La commande INSERT

La commande **INSERT** permet d'ajouter de nouvelles lignes à une table

```
INSERT  
INTO      table [ (liste de colonnes) ]  
{VALUES (liste de valeurs) | requête}
```

- Dans le cas où la liste de colonnes n'est pas spécifiée tous les attributs de la table cible doivent être fournis dans l'ordre de déclaration
- Si seulement certaines colonnes sont spécifiées, les autres sont insérées avec la valeur NULL
- Une insertion à partir d'une requête permet d'insérer plusieurs lignes dans la table cible à partir d'une autre table

- **Insertion d'une seule ligne**

Q31 Ajouter le client ('c100', 'Duduche', 'Nice') dans la table client

```
INSERT
INTO      client (IdCli, nom, ville)
VALUES    ('c100', 'Duduche', 'Nice')
```

- **Insertion de plusieurs lignes**

Q32 Ajouter dans une table « temp » de même schéma que la table Vente, toutes les ventes qui sont antérieures au 01-Jan-1994

```
INSERT
INTO      temp (IdCli, IdPro, date, qte)
          SELECT V.no_cli, V.IdPro, V.date, V.qte
FROM      vente V
WHERE     V.date < '01-jan-1994'
```

## □ La commande UPDATE

La commande **UPDATE** permet de changer des valeurs d'attributs de lignes existantes

UPDATE	table
SET	liste d'affectations
[ WHERE	qualification ]

- L'absence de clause WHERE signifie que les changements doivent être appliqués à toutes les lignes de la table cible
- Exemples

Q33 Augmenter de 20% les prix de tous les produits

UPDATE	produit
SET	prix = prix * 1.2

Q34 Augmenter de 50% les prix des produits achetés par des clients de Nice

UPDATE	produit
SET	prix = prix * 1.5
WHERE	EXISTS
	(
	SELECT *
	FROM vente V , client C
	WHERE V.IdCli = C.IdCli
	AND C.ville = 'Nice'
	)



## ▣ La commande DELETE

La commande **DELETE** permet d'enlever des lignes dans une table

```
DELETE
FROM      table
[ WHERE   qualification ]
```

- L'absence de clause WHERE signifie que toutes les lignes de la table cible sont enlevées

### ▪ Exemples

Q35 Supprimer les ventes antérieures au 01-jan-1994

```
DELETE
FROM      vente
WHERE     date < '01-jan-1994'
```

Q36 Supprimer les ventes des clients de Nice antérieures au 01-mar-1994

```
DELETE
FROM      vente
WHERE     date < '01-mar-1994'
AND IdCli IN
(
    SELECT C.IdCli
    FROM   client C
    WHERE  C.ville = 'Nice'
)
```

## 4 Contrôle des données

### ▣ Contrôle des accès concurrents

#### La notion de transaction

Une *transaction* est une unité logique de traitement qui est soit complètement exécutée, soit complètement abandonnée

- Une transaction fait passer la BD d'un état cohérent à un autre état cohérent
- Une transaction est terminée
  - soit par **COMMIT**
  - soit par **ROLLBACK**

## □ La commande COMMIT

La commande **COMMIT** termine une transaction avec succès ; toutes les mises à jour de la transaction sont validées

- On dit que la transaction est *validée*
- Tous ses effets sont alors connus des autres transactions s'exécutant concurremment

## □ La commande ROLLBACK

La commande **ROLLBACK** termine une transaction avec échec ; toutes les mises à jour de la transaction sont annulées (tout se passe comme si la transaction n'avait jamais existé)

- On dit que la transaction est *annulée*
- Aucune des opérations effectuées par cette transaction n'est connue des autres transactions

## ▣ Contrôle des droits d'accès

Chaque créateur d'une table est **propriétaire** de cette table et obtient tous les droits d'accès à cette table (i.e. les droits d'effectuer les opérations SELECT, INSERT, UPDATE, DELETE)

- Le propriétaire d'une table peut passer ses privilèges sélectivement à d'autres utilisateurs ou à tout le monde (**PUBLIC**)

## ▣ La commande GRANT

La commande **GRANT** permet de passer des droits d'accès à un utilisateur ou un groupe d'utilisateurs

**GRANT** privilèges **ON** table **TO** bénéficiaire  
**[WITH GRANT OPTION]**

- Les privilèges qui peuvent être passés sont :
  - soit **ALL** (tous les privilèges)
  - soit une liste de privilèges parmi :
    - **SELECT**
    - **INSERT**
    - **UPDATE [(liste de colonnes)]**  
l'omission de la liste de colonnes signifie toutes les colonnes
    - **DELETE**

- Le bénéficiaire peut être :
  - soit **PUBLIC** (tous les utilisateurs)
  - soit un utilisateur ou un groupe d'utilisateurs
- L'option **WITH GRANT OPTION** permet de passer un privilège avec le droit de le transmettre
- Exemples

```
GRANT SELECT ON produit TO PUBLIC  
GRANT INSERT, UPDATE ON produit TO toto
```

- Aucun utilisateur ne peut passer un privilège qu'il ne détient pas

## □ La commande **REVOKE**

La commande **REVOKE** permet de retirer des droits à un utilisateur ou groupe d'utilisateurs

```
REVOKE privilèges ON table FROM bénéficiaire
```

## 5 Les vues

Une **vue** est une table virtuelle calculée à partir des tables de base par une requête

- Une vue apparaît à l'utilisateur comme une table réelle, cependant les lignes d'une vue ne sont pas stockées dans la BD (uniquement sa définition est enregistrée dans le DD)
- Les vues assurent **l'indépendance logique**
- Elles peuvent être utilisées pour cacher des données sensibles, ou pour montrer des données statistiques  
Ex.:

```
CREATE VIEW      prix-caché AS
      SELECT     P.IdPro, P.nom, P.marque
      FROM       produit P
```

```
CREATE VIEW      stat-vente ( IdPro, tot-qte )
      AS SELECT   V.IdPro, SUM ( V.qte )
      FROM       vente V
      GROUP BY   V.IdPro
```

## □ La commande CREATE VIEW

La commande **CREATE VIEW** crée la définition d'une vue

```
CREATE VIEW vue [(liste de colonnes)]  
AS requête [ WITH CHECK OPTION ]
```

### ■ Ex.:

```
CREATE VIEW      produitIBM ( no, nom, prx )  
      AS SELECT  P.IdPro, P.nom, P.prix  
      FROM      produit P  
      WHERE     P.marque = 'IBM'
```

- Les données des tables de bases peuvent être modifiées dans certains cas au travers d'une vue, mais cela n'est pas toujours possible
- L'option **WITH CHECK OPTION** permet de vérifier que les lignes insérées dans une table de base au-travers d'une vue vérifient les conditions exprimées dans la requête. Cela permet d'imposer des contraintes d'intégrité lors des mises à jour au travers de la vue

## ▣ Intérêt des vues

### ▪ Indépendance logique

Le concept de vue permet d'assurer une indépendance des applications vis-à-vis des modifications du schéma

### ▪ Simplification d'accès

Les vues simplifient l'accès aux données en permettant par exemple une pré-définition des jointures et en masquant ainsi à l'utilisateur l'existence de plusieurs tables

Ex. :

La vue qui calcule les moyennes générales pourra être consulté par la requête :

```
SELECT * FROM Moyennes
```

### ▪ Confidentialité des données

Une vue permet d'éliminer des lignes sensibles et/ou des colonnes sensibles dans une table de base





Ce chapitre aborde les techniques mises en œuvre par les SGBD pour traiter les problèmes de concurrence d'accès, de reprise après panne, de sécurité, d'intégrité, ainsi que des méthodes d'accès aux données.

## **I GESTION DE TRANSACTIONS**

## **II REPRISE APRÈS PANNE**

## **III SÉCURITÉ**

## **IV INTÉGRITÉ**

## **V MÉTHODES D'ACCÈS**

# I Gestion de transactions

## ▣ La notion de transaction

Unité logique de traitement qui est :

- soit complètement exécutée
- soit complètement abandonnée

- ❖ Une transaction est une **unité atomique** de traitement
- ❖ Une transaction fait passer la base de données d'un état **cohérent** à un autre état cohérent
- ❖ Si une transaction ne va pas à son terme pour une raison ou pour une autre, la base est restaurée dans l'état où elle se trouvait avant que la transaction ne démarre

Exemple du banquier :

le transfert d'une somme  $S$  d'un compte  $C1$  vers un compte  $C2$

(1) début-transaction

(2) lire  $C1$

(3)  $C1 := C1 - S$

(4) écrire  $C1$

(5) lire  $C2$

(6)  $C2 := C2 + S$

(7) écrire  $C2$

(8) fin-transaction

Cette transaction est constituée d'un ensemble d'actions élémentaires, mais elle doit être traitée comme une seule opération.

Autrement dit le gestionnaire des transactions doit assurer que **toutes** les actions de la transaction sont exécutées, ou bien qu'**aucune** ne l'est.

## □ La vie d'une transaction

### ◆ Vie sans histoire

La transaction s'exécute normalement jusqu'à la fin. Elle se termine par une instruction de **validation** **COMMIT** en SQL

Nous dirons que cette transaction est **validée**. Toutes les modifications faites sur la base par cette transaction sont considérées comme définitives.

### ◆ Un assassinat

Un événement extérieur vient interrompre l'exécution de la transaction de façon irrémédiable

Cet arrêt peut provenir, soit d'une panne, soit d'une action délibérée de la part du SGBD qui décide de supprimer telle ou telle transaction (c'est le cas lorsqu'il détecte un interblocage).

### ◆ Un suicide

Au cours de son exécution la transaction détecte certaines conditions qui font que la poursuite de son exécution s'avère impossible, elle peut se supprimer en exécutant une instruction **d'annulation** **ROLLBACK** en SQL

- ❖ Dans ces deux derniers cas, tout doit se passer comme si la transaction n'avait jamais existée. Il faut donc en quelque sorte lui faire faire marche arrière et effacer de la base de données toute trace de son exécution : nous dirons que la transaction a été **annulée**.

## ▣ La gestion des transactions

Un système de gestion transactionnel doit garantir les propriétés suivantes (résumées par le vocable **ACID**) :

### **Atomicité**

Une transaction doit effectuer toutes ses mises à jour ou rien faire du tout

### **Cohérence**

La transaction doit faire passer la base de données d'un état cohérent à un autre

### **Isolation**

Les résultats d'une transaction ne doivent être visibles aux autres transactions qu'une fois la transaction validée

### **Durabilité**

Dès qu'une transaction valide ses modifications, le système doit garantir que ces modifications seront conservées en cas de panne

## ▣ Les problèmes de concurrence d'accès

Des transactions exécutées concurremment peuvent interférer et mettre la base de données dans un état incohérent.

Considérons deux transactions T1 et T2 qui s'intéressent à un même objet A

Les deux seules opérations possibles sur A, sont :  
**lire** et **écrire**

Quatre possibilités :

### 1. LECTURE-LECTURE ET PARTAGE

- Aucun conflit
- un même objet peut toujours être partagé en lecture

## 2. ECRITURE-ECRITURE ET PERTE DE MISE A JOUR

T2 vient "écraser" par une autre écriture celle effectuée par T1

Temps	Transaction T1	Etat de la base	Transaction T2
t1	lire A	A = 10	-
t2	-		lire A
t3	A := A + 10		-
t4	-		-
t5	-		A := A + 50
t6	écrire A	A = 20	-
t7	-	A = 60	écrire A



### 3. ECRITURE-LECTURE ET LECTURES IMPROPRES

T2 lit une valeur modifiée par T1 et ensuite T1 est annulée

Temps	Transaction T1	Etat de la base	Transaction T2
t1	lire A	A=10	-
t2	A := A+ 20		-
t3	écrire A	A = 30	-
t4	-		lire A
t5	**annulation**		
t6	-		-

#### 4. LECTURE-ECRITURE ET LECTURES NON REPRODUCTIBLES

T1 modifie la valeur de A entre deux lectures de T2

Temps	Transaction T1	Etat de la base	Transaction T2
t1	lire A	A=10	-
t2	-		lire A
t3	A := A + 10		-
t4	écrire A	A = 20	-
t5	-		-
t6	-		lire A

- De nombreuses solutions ont été proposées pour traiter le problème des accès concurrents.
- Un exemple important est le protocole appelé *verrouillage à deux phases* qui est un des plus utilisés.

## ▣ Verrouillage

Il repose sur les deux actions :

verrouiller (A) : acquérir un contrôle de l'objet A

libérer (A) : libérer l'objet A

❖ Un objet A est typiquement un n-uplet de la BD

### Il y a deux types de verrous :

◆ Verrous **exclusifs** (X locks)  
ou verrous d'écriture

◆ Verrous **partagés** (S locks)  
ou verrous de lecture

## **Protocole d'accès aux données**

1. Aucune transaction ne peut effectuer une lecture ou une mise à jour d'un objet si elle n'a pas acquis au préalable un verrou S ou X sur cet objet
2. Si une transaction ne peut obtenir un verrou déjà détenu par une autre transaction T2, alors elle doit attendre jusqu'à ce que le verrou soit libéré par T2
3. Les verrous X sont conservés jusqu'à la fin de la transaction (COMMIT ou ROLLBACK)
4. En général les verrous S sont également conservés jusqu'à cette date

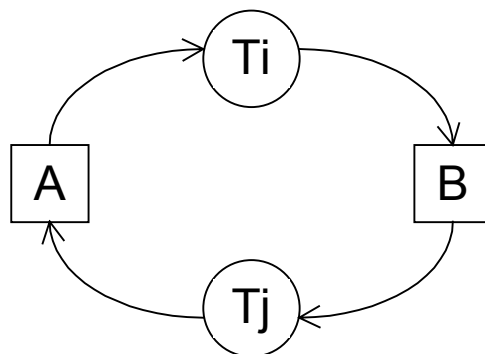
## Phénomènes indésirables

### *La privation*

Une transaction risque d'attendre un objet indéfiniment si à chaque fois que cet objet est libéré, il est pris par une autre transaction.

Pour traiter ce problème, on peut organiser sur chaque verrou une file d'attente avec une politique "première arrivée", "première servie".

### *L'interblocage (ou verrou mortel)*



Ti attend Tj , Tj attend Ti : il y a interblocage

On peut construire le graphe "qui attend quoi" :

- les sommets représentent les transactions  $T_i$
- on aura une arête  $T_i \rightarrow T_j$  si  $T_i$  est en attente de  $T_j$

Il y a situation d'interblocage lorsque le graphe contient un cycle.

2 techniques pour traiter le problème d'interblocage :

## ❖ **Prévention des interblocages**

Lorsqu'une demande d'acquisition de verrou ne peut être satisfaite on fait passer un test aux deux transactions impliquées, à savoir celle qui demande le verrou,  $T_i$ , et celle qui le possède déjà,  $T_j$ .

Si  $T_i$  et  $T_j$  passent le test alors  $T_i$  est autorisé à attendre  $T_j$ , sinon l'une des deux transactions est annulée pour être relancée par la suite.

## ❖ **Détection des interblocages**

Les interblocages sont détectés en construisant effectivement le graphe "qui attend quoi" et en y recherchant les cycles.

Lorsqu'un cycle est découvert l'une des transactions est choisie comme victime, elle est annulée de manière à faire disparaître le cycle.

L'examen du graphe peut se faire

- soit lorsqu'il y a attente de la part d'une transaction,
- soit périodiquement

## □ Sériabilité

Une exécution entrelacée donnée d'un ensemble de transactions est considérée correcte si elle est sérialisable – c'est-à-dire, si elle produit le même résultat qu'une certaine exécution *en série* des mêmes transactions s'exécutant l'une après l'autre

## Ordonnement

Etant donné un ensemble de transactions, toute exécution de ces transactions (entrelacée ou non) est appelé un ordonnancement

## Théorème de verrouillage à deux phases :

Si toutes les transactions satisfont le « *protocole de verrouillage à deux phases* », tous les ordonnancements entrelacés sont alors sérialisables.

## □ **Protocole de verrouillage à deux phases**

Pour chaque transaction, tous les verrouillages doivent précéder toutes les libérations de verrous.

Après l'abandon d'un verrou, une transaction ne doit plus jamais pouvoir obtenir de verrous

On distingue deux phases :

- acquisition des verrous
  - libération des verrous
- 
- Dans la pratique, la seconde phase est souvent condensée en une seule opération de COMMIT ou de ROLLBACK à la fin de la transaction.
  - Dans le but de réduire les conflits sur les ressources et, par la même, d'améliorer les performances, les systèmes réels autorisent la construction de transactions qui ne sont pas à deux phases – c'est-à-dire qui abandonnent prématurément des verrous (avant le COMMIT) et obtiennent ensuite de nouveaux verrous.



## ▣ Niveaux d'isolation

Tout protocole qui n'est pas complètement sérialisable ne peut être considéré sûr, cependant, les systèmes autorisent des transactions s'exécutant à un **niveau d'isolation** non sûr qui pourrait violer la sérialisabilité de trois façons particulières :

### ❖ Lecture salissante

Supposons que la transaction T1 effectue une mise à jour sur une certaine ligne, que la transaction T2 récupère ensuite cette ligne et que la transaction T1 se termine par un ROLLBACK. La transaction T2 a alors observé une ligne qui n'existe plus, et dans un certain sens n'a jamais existé (car la transaction T1 n'a en fait jamais été exécutée).

### ❖ Lecture non renouvelable

Supposons que la transaction T1 récupère une ligne, que la transaction T2 effectue ensuite une mise à jour de cette ligne et que la transaction T1 récupère de nouveau la « même » ligne. La transaction T1 a en fait récupéré la « même » ligne deux fois mais a observé des valeurs différentes de cette ligne.

### ❖ Fantômes

Supposons que la transaction T1 récupère un ensemble de lignes qui satisfont une certaine condition. Supposons que la transaction T2 insère ensuite une ligne qui satisfait la même condition. Si la transaction T1 répète maintenant la même demande, elle observera une ligne qui n'existait pas précédemment – un « fantôme ».

## Niveaux d'isolation SQL

L'instruction **SET TRANSACTION** permet de définir le *niveau d'isolation* de la prochaine transaction à exécuter

**SET TRANSACTION READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE**

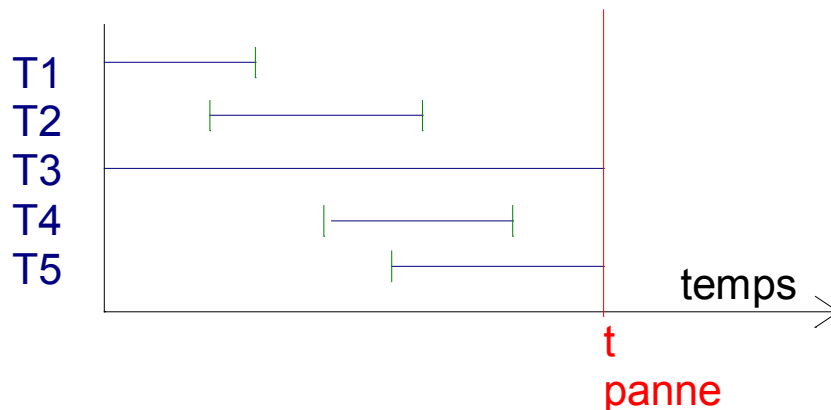
- L'option par défaut est **SERIALIZABLE**

Niveau d'isolation	lecture salissante	lecture non renouvelable	fantôme
<b>READ UNCOMMITTED</b> Lecture non validée	O	O	O
<b>READ COMMITTED</b> Lecture validée	N	O	O
<b>REPEATABLE READ</b> Lecture renouvelable	N	N	O
<b>SERIALIZABLE</b> Sérialisable	N	N	N

## II La reprise après panne

### □ Le problème des pannes

Analysons un exemple :



Une panne se produit au temps  $t$

Les deux transactions T3 et T5 sont en cours d'exécution  
Les transactions T1, T2 et T4 s'étaient terminées  
correctement avant la panne

Les effets de T1, T2 et T4 doivent survivre à la panne  
Ceux de T3 et T5 doivent être éliminés.

Que faut-il faire ?

- défaire le travail de T3 et T5
- refaire, totalement ou partiellement, le travail de T1, T2 et T4 à partir d'un état de la base conservé au préalable  
(bien que l'on ait gardé une trace du fait que T1, T2 et T4 ont atteint leur point de confirmation, il pourra se faire que la panne a détruit les modifications effectuées par T1, T2 et T4 sur la base)

## □ Journalisation des transactions

L'activité des transactions est enregistrée dans un *journal*. géré par le SGBD

Nous supposons que ce journal n'est jamais détruit totalement, par exemple en gérant deux copies du même journal.

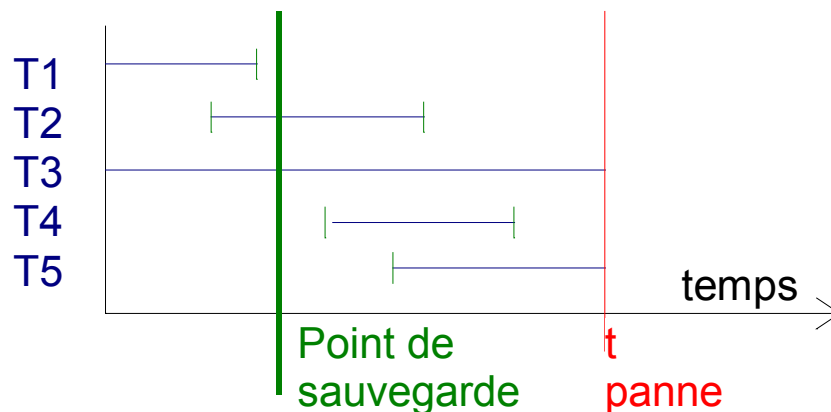
Ce journal est composé d'articles qui indiquent les événements principaux qui affectent la base de données :

- début d'une nouvelle transaction,
- fin d'une transaction et donc confirmation de ses mises à jour,
- annulation d'une transaction,
- pour une mise à jour on conserve :
  - 1) l'identificateur de la transaction effectuant la maj,
  - 2) l'identificateur du n-uplet modifié,
  - 3) l'ancienne valeur,
  - 4) la nouvelle valeur affectée.

De plus des *sauvegardes* de la base sont faites à intervalles réguliers (sur bande ou sur d'autres disques). Un tel événement est enregistré dans le journal.

## Redémarrage du SGBD

Nous supposons qu'il y a eu une sauvegarde totale de la base au temps indiqué ci-dessous :



Pour remettre la base dans un état cohérent, le SGBD doit :

- refaire le travail des transactions ayant été confirmées avant la panne (ici T1, T2 et T4)
- défaire le travail de celles qui étaient actives au moment de la panne (ici T3 et T5)

Aucune des modifications de T5 n'apparaît dans la sauvegarde donc T5 est déjà défaite.

Toutes les modifications faites par T1 figurent dans la sauvegarde, donc T1 n'est pas à refaire.

Refaire T2 et T4 nécessite de parcourir le journal vers l'avant à partir du point de sauvegarde. Puisque le journal contient les nouvelles valeurs, il suffit de refaire chacune des modifications (toutes les modifications faites avant la sauvegarde figurent déjà dans la base).

Défaire T3 nécessite un parcours à reculons du journal : pour chaque modification faites, il faut revenir à l'ancienne valeur et ceci jusqu'à ce qu'on rencontre le début-transaction de T3.

## III Sécurité

### Protection des données contre les accès non autorisés

#### ▣ Contrôle d'accès ou autorisation

Les contrôles d'accès vérifient l'identité des usagers qui se présentent et en conséquence leur assignent des droits d'accès sur tel ou tel ensemble de données.

#### Autorisation (*GRANT* en SQL)

Tout usager qui a le droit de transmettre des privilèges sur un objet peut utiliser la commande *GRANT* pour transmettre ce privilège :

***GRANT*** privilèges **ON** objet **TO** liste d'usagers  
**[WITH GRANT OPTION]**

- Les privilèges peuvent être :
  - lire (*SELECT*),
  - insérer de nouveaux n-uplets (*INSERT*),
  - modifier des valeurs (*UPDATE*),
  - supprimer la totalité d'une relation (*DROP*),
  - créer de nouvelles relations (*CREATE*).
- L'option facultative *WITH GRANT OPTION* permet au donneur d'autoriser le receveur à transmettre à d'autres les privilèges qu'il reçoit.
- Un usager peut recevoir un privilège de plusieurs sources différentes.

## Révocation (*REVOKE* en SQL)

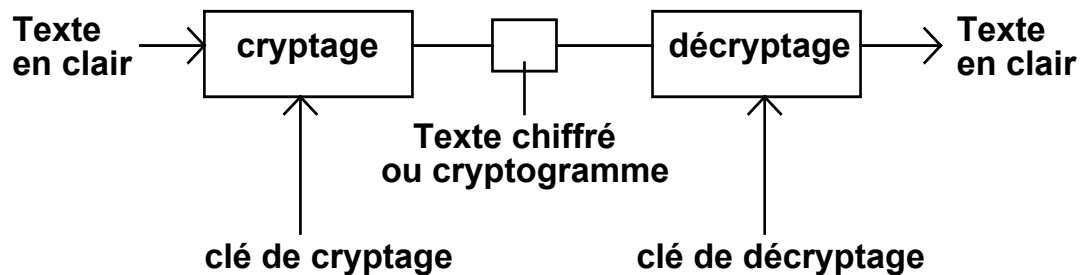
Tout usager ayant donné un privilège peut à tout moment retirer ce privilège grâce à la commande *REVOKE* :

**REVOKE** privilèges **ON** objet **FROM** liste d'usagers

- Les privilèges sur l'objet mentionné sont retirés au receveur à moins que ce dernier ne les ait reçus d'une autre source, indépendante.
- Cette procédure de révocation complique le mécanisme d'autorisation car il faut appliquer récursivement les procédures de révocation puisqu'un usager auquel on retire un privilège a pu le transmettre à d'autres.

## □ Cryptographie

La cryptographie a pour but de stocker ou de transporter l'information sous une forme telle que seuls les usagers en possession de la clé de décryptage sont susceptibles de la comprendre.



## Cryptographie à clé publique

Elle fait appel à 2 clés

- une clé privée (gardée secrète par son détenteur) qui ne sert qu'au décryptage
- une clé publique qui n'est utilisée que pour crypter

L'algorithme de cryptage  $C$  et l'algorithme de décryptage  $D$  sont choisis de telle sorte que le calcul de  $D$  soit très complexe même si l'on connaît complètement  $C$

Ex.: Paul souhaite envoyer le message  $M$  à Jacques.

- Paul utilise la clé publique  $C_{\text{Jacques}}$  de Jacques pour crypter le message qu'il transmet à Jacques.
- Jacques déchiffre le message reçu en lui appliquant  $D_{\text{Jacques}}(C_{\text{Jacques}}(M))$ , personne d'autre n'est capable de déchiffrer le message  $C_{\text{Jacques}}(M)$ .



## L'algorithme du MIT

1. choisir 2 nombres premiers, p et q, chacun plus grands que  $10^{100}$
2. calculer  $n=p.q$  et  $z=(p-1)(q-1)$
3. choisir un nombre d premier avec z
4. chercher un nombre e tel que  $e.d=1(\text{mod } z)$

Découper le texte en une suite de blocs de telle sorte que chaque bloc de texte en clair M soit un nombre tel que  $0 \leq M < n$

pour crypter :  $C = M^e(\text{mod } n)$  la clé publique = (e,n)

pour déchiffrer :  $D = C^d(\text{mod } n)$  la clé privée = d

La sécurité de la méthode réside dans la difficulté à décomposer de très grands nombres en facteurs premiers.

Ex.:

$p=3, q=11, n=33, z=20, d=7, e=3$

Texte en clair	N	I	C	E
M	14	9	3	5
$M^3$	2744	729	27	125
Texte chiffré $C=M^3(\text{mod } 33)$	5	3	27	26
$C^7$	78125	2187	-	-
$C^7(\text{mod } 33)$	14	9	3	5
Texte en clair	N	I	C	E

## IV Intégrité

### Contrôle de la validité des données

#### ▣ **Contrainte d'intégrité**

Une contrainte d'intégrité est une assertion qui doit être vérifiée par des données à des instants déterminés.

- Les contraintes d'intégrité permettent de préciser davantage la partie intentionnelle (sémantique) de la base de données.
- Une base de données est cohérente vis à vis des contraintes qui sont exprimées, si ces contraintes sont respectées par les données de la base.

## ▣ **Gestion des contraintes d'intégrité**

### **Expression des contraintes**

L'écriture des différents types de CI est prévue dans de nombreux langages

Par exemple la clause CHECK de SQL/ORACLE

### **Vérification des contraintes**

Les CI sont vérifiées lors des mises à jour (en fin de transaction)

C'est très coûteux en temps machine, il est essentiel de pouvoir vérifier ces contraintes de manière efficace

### **Violation des contraintes**

Une mise à jour qui provoque la violation d'une CI est refusée

L'intégrité de la base de données est préservée par le SGBD

## ▣ Utilisation des déclencheurs (trigger)

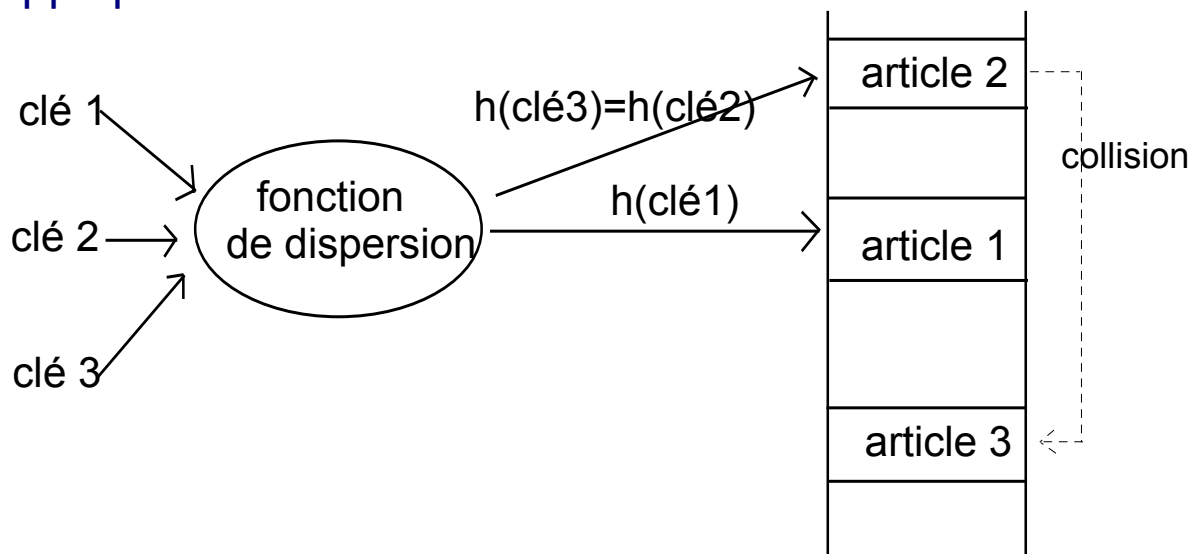
Un **déclencheur** (*trigger*) permet de définir un ensemble d'actions qui sont déclenchées automatiquement par le SGBD lorsque des mises à jour sont effectuées.

- Les actions sont enregistrées dans la base et plus dans les programmes d'application
- Cette notion n'est pas encore spécifiée dans SQL 2
- Elle est présente dans les principaux SGBD (Oracle, Sybase, DB2, SQL Server)

## V Méthodes d'accès

### ❑ Méthode d'accès par dispersion

L'adresse relative d'un article (ou d'un paquet contenant l'article) est obtenu par une fonction de hachage appliquée à la clé



### Traitement des collisions lorsqu'un paquet est plein

- **adressage ouvert**  
prendre le premier paquet suivant ayant de la place libre
- **chaînage**  
chaîner un paquet de débordement au paquet plein
- **rehachage**  
appliquer une deuxième fonction de hachage

### Avantages

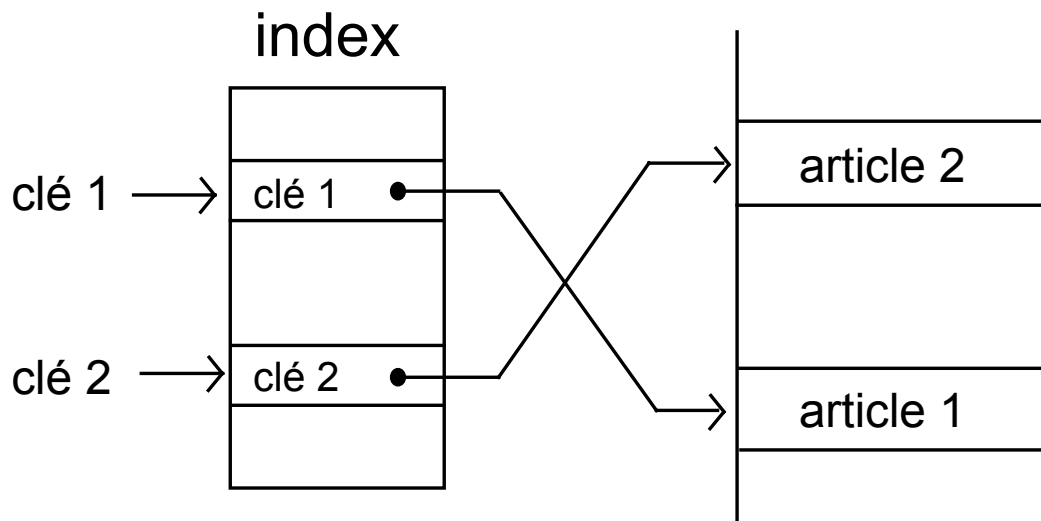
rapidité si l'on parvient à réaliser un faible taux de collisions

### Inconvénients

pas d'accès séquentiel trié dans l'ordre des clés

## □ Méthodes d'accès par indexage

L'adresse relative d'un article (ou d'un paquet contenant l'article) est recherchée à partir de la clé dans une table d'index



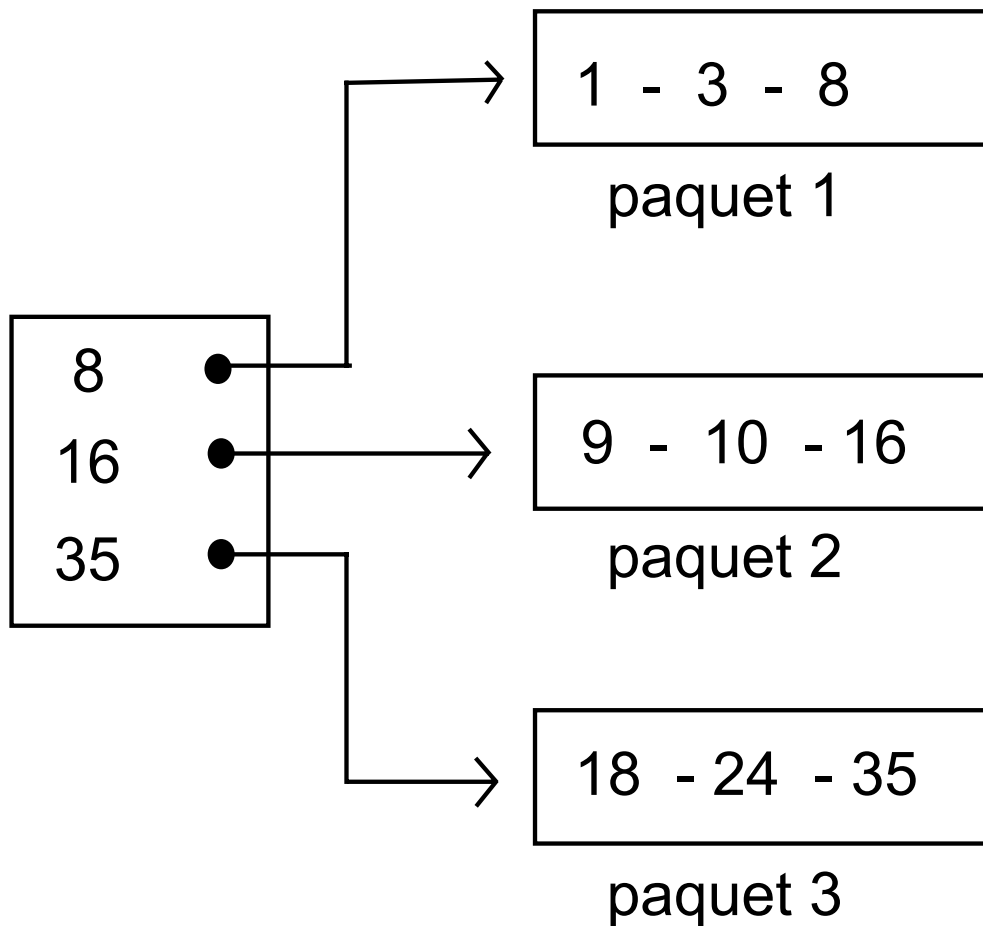
Principales méthodes d'accès indexées

**ISAM**

**ARBRE B**

elles se distinguent par le mode de placement des articles et par l'organisation de l'index

## ISAM (Indexed Sequential Access Method d'IBM)

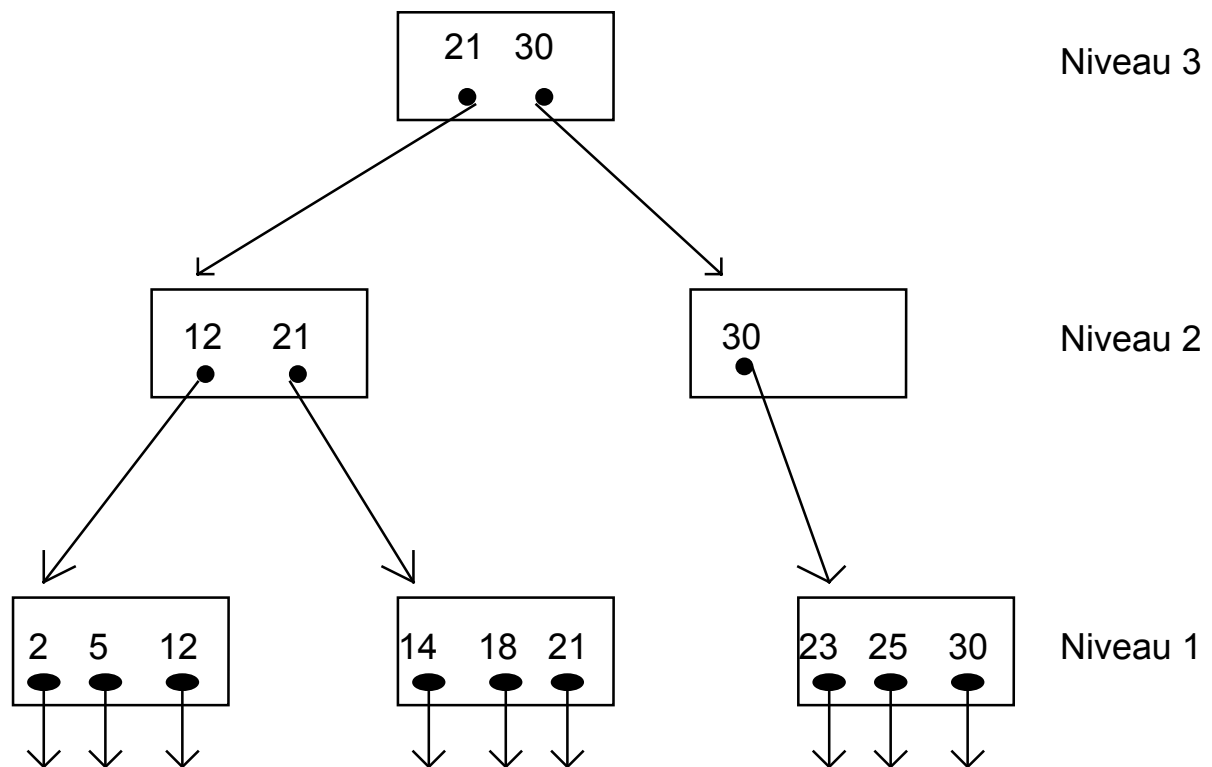


Les articles sont rangés dans des paquets de taille fixe par ordre croissant des clés

Chaque paquet correspond à une entrée en index contenant le doublet :  
(plus grande clé du paquet, adresse relative du paquet)

## ARBRE B

L'index est hiérarchisé en plusieurs niveaux



L'index est composé de paquets de clés

2 types de pointeurs :

- pointeur interne permettant de représenter l'arbre
- pointeur externe sur l'adresse relative d'un article

Les paquets de niveau  $k+1$  contiennent les plus grandes clés des paquets de niveau  $k$



# **Chapitre 7      L'architecture client serveur et ODBC**

## **I- L'architecture client serveur**

## **II- Les normes du client serveur**

## **III- ODBC**

1. Généralités
2. Composants ODBC
3. Niveaux de conformité
4. Structure d'un programme ODBC
5. Fonctions de connexion et de déconnexion
6. Transactions dans ODBC
7. Envoi de requêtes SQL
8. Récupération des résultats
9. Détection des erreurs
10. Exemples d'application ODBC

## I L'architecture client serveur

L'architecture client serveur est caractérisée par une répartition des programmes entre processus *client* et processus *serveur* communiquant sur un réseau.



### Dialogue client serveur

- Le **client** est le processus qui adresse à un serveur une requête spécifique correspondant à une demande de service.
- Le **serveur** est un processus à l'écoute d'une demande de service en provenance d'un processus client.

## ▣ Serveur de bases de données

- Ils offrent des services classiques d'accès à une base de données :
  - ❖ recherches, mises à jour
  - ❖ confidentialité, intégrité
  - ❖ traitement des transactions
  - ❖ reprise après panne
  
- Ils exécutent des requêtes SQL et des procédures stockées pour le compte du client
  
- Aujourd'hui, basés sur des SGBD relationnels

## ▣ Le middleware

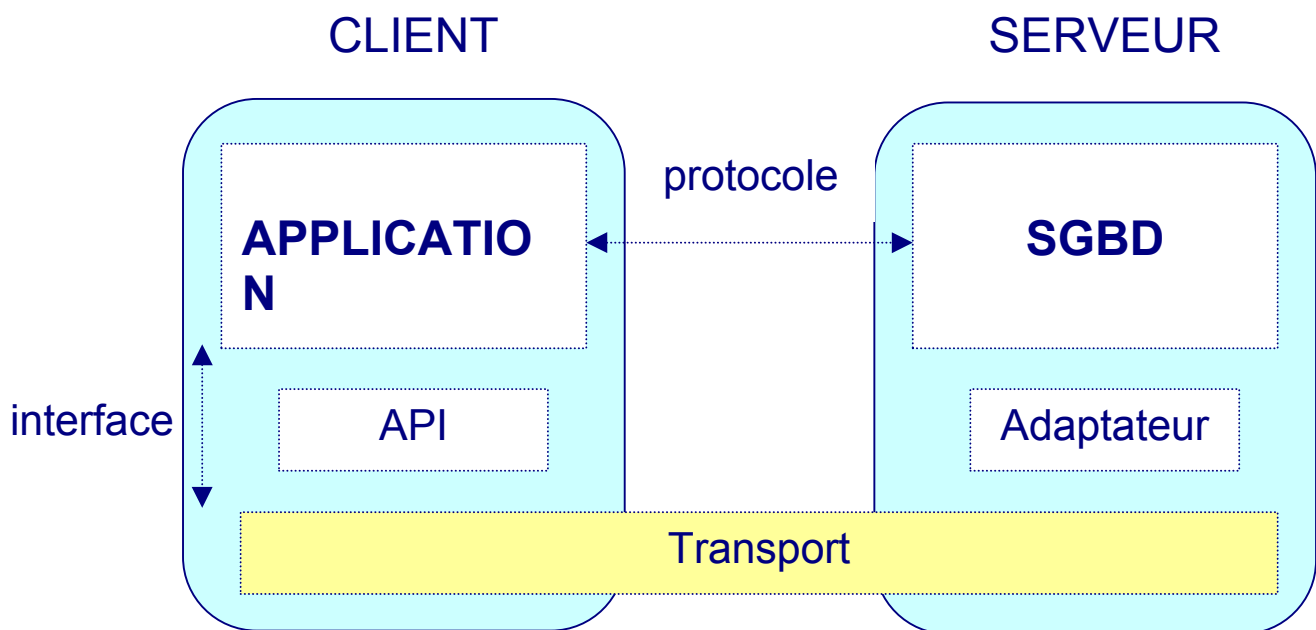
- Le *middleware* est ce logiciel du milieu qui assure les dialogues entre clients et serveurs souvent hétérogènes.
- Ensemble des services logiciels construits au-dessus d'un protocole de transport afin de permettre l'échange de requêtes et des réponses associées entre client et serveur de manière transparente.

## ▣ Les systèmes ouverts

- Un *système ouvert* est un système dont les interfaces obéissent à des **standards internationaux** établis au sein de structures accessibles à tous.
- De nombreux groupes proposent des standards, dont l'**ISO**, l'**ANSI**, le **CCITT**, l'**IEEE**.

## ▣ **API (*Application Programming Interface*)**

- Bibliothèque de fonctions permettant de développer des applications client serveur
- Les programmes clients envoient leurs requêtes au serveur par des appels à des fonctions contenues dans l'API



### **Interface applicative**

## ▣ API propriétaire du SGBD

- fourni par l'éditeur du SGBD
- permet uniquement l'accès à la base pour laquelle elle a été développée

## Exemples

❖ OCI d'Oracle

❖ DB-Lib de Sybase

❖ SQL/Services de RDB

## ▣ API indépendante du SGBD

- fourni par un constructeur indépendant du SGBD permet l'accès à des SGBD différents

### Exemples

❖ ODBC de Microsoft

❖ IDAPI de Borland, Novell et IBM



## ▣ Les principaux serveurs SQL

❖ ORACLE

❖ DB2

❖ SYBASE

❖ CA-OPEN INGRES

❖ INFORMIX

❖ SQL SERVER

## II Les normes du client serveur

### ▣ CLI (*Call Level Interface*)

- interface applicative SQL
- interface unique permettant l'accès à des SGBDR différents
- travaux du SAG (SQL Access Group)
- standard X/Open

## ▣ **RDA** (*Remote Data Access*)

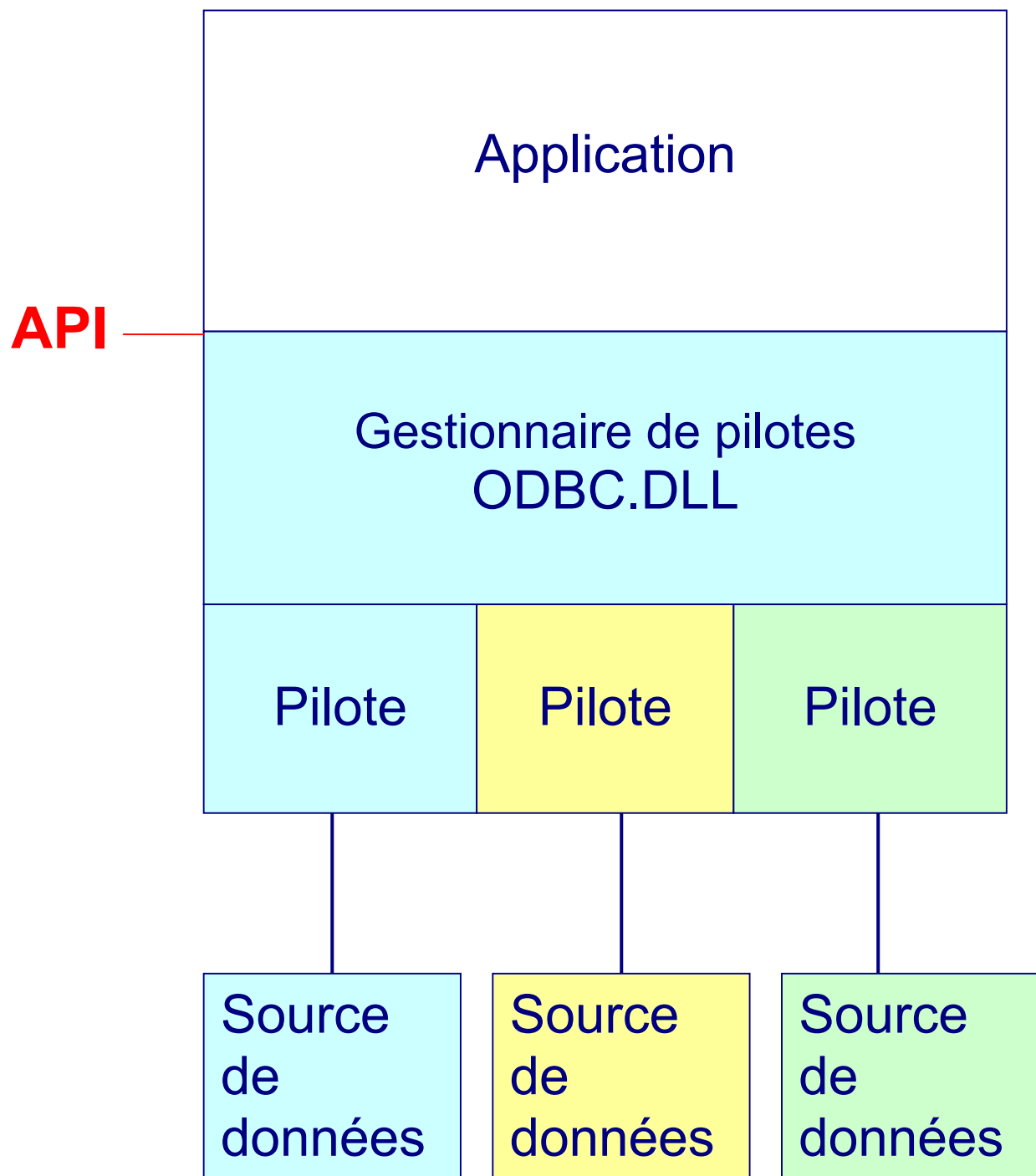
- protocole d'application construit au-dessus des couches présentation et session de l'architecture OSI de l'ISO
- les messages permettent le transport des requêtes générées par l'interface CLI et les réponses associées
- standard ISO

## III ODBC

### 1. Généralités

- Open Data Base Connectivity
- Implémentation du standard CLI
- Accès normalisé à des SGBD relationnels différents (Oracle, DB2 ...)
- Accès même à des pseudo-SGBD, ou des tableurs, ou encore des gestionnaires de fichiers
- Interopérabilité avec des sources de données hétérogènes
- Avec ODBC, il est possible de développer une application sans se soucier de la source de données qui sera utilisée en exploitation
- **API C** (SDK ODBC) et classes C++ (MFC)

## 2. Composants ODBC



### Composants ODBC

# Application

- Connexion à un SGBD
- Envoi de requêtes SQL
- Récupération des résultats
- Gestion des erreurs
- Gestion des transactions
- Déconnexion

# Gestionnaire de pilotes

- Charge dynamiquement les pilotes correspondant aux sources de données auxquelles l'application souhaite se connecter
- Consulte le fichier **ODBC.INI** pour retrouver le pilote
- Transmet les requêtes au pilote
- Transmet les résultats à l'application
- Pour accéder à un nouveau SGBD, il suffit d'installer un pilote spécifique à ce SGBD (aucun changement dans l'application)
- Une application peut établir plusieurs connexions à différentes sources de données

# Fichier ODBC.INI

- Définit des sources de données
- Exemple

```
[ODBC Data Sources]
iut1=Oracle73 Ver 2.5 (32 bit)
...
[iut1]
Driver32=C:\ORANT\ODBC250\sqo32_73.dll
```

- La section **[ODBC Data Sources]** donne le nom de chaque source disponible et le pilote associé
- A chaque source correspond une section particulière donnant des informations supplémentaires : le nom du serveur, le protocole utilisé pour les communications ...



# Administrateur ODBC

- ajoute les sources de données dans le fichier ODBC.INI en utilisant l'utilitaire *ODBC Administrator*
- installe les pilotes ODBC
- établit les connexions avec des BD physiques

# Pilote

- Deux types de pilotes
  - Pilotes traitants (single-tier)
    - ❖ traitent les requêtes SQL
    - ❖ destinés à des BD non-SQL
    - ❖ analyse, traduit les instructions SQL en opérations élémentaires de fichier et les transmet à la source de données
  - Pilotes transparents (multiple-tier)
    - ❖ transmettent les requêtes SQL à un serveur qui les traitent

# Source de données

- Données auxquelles un utilisateur souhaite accéder
- Identifiée par une entrée dans le fichier ODBC.INI
- Chaque entrée de nom de source dans ODBC.INI spécifie des informations de connexion

### 3. Niveaux de conformité

- En principe, une application ODBC devrait pouvoir interopérer avec n'importe quelle source de données.
- Mais en pratique, les pilotes et les sources de données associées n'offrent pas tous les mêmes possibilités de fonctionnalités de l'API et de requêtes SQL
- Niveaux de conformité API
  - Définit différents niveaux de fonctions de l'API
  - Un pilote particulier précise son niveau de conformité API
- Niveaux de conformité SQL
  - Définit différents niveaux de grammaire SQL

# Niveaux de conformité API

- Le niveau noyau (Core API)
  - ❖ Correspond au standard CLI de l'X/Open
  - ❖ Allocation et libération de descripteurs d'environnement, de connexion et d'instruction
  - ❖ Fonction de connexion
  - ❖ Préparation et exécution d'instruction SQL
  - ❖ Exécution directe d'instructions SQL
  - ❖ Liaison pour des paramètres SQL et des colonnes de résultats
  - ❖ Validation ou annulation de transactions
  - ❖ Récupération d'informations sur des erreurs

## Niveaux de conformité API

- Le niveau 1 (Level 1 API)

- ❖ Noyau +

- ❖ Fonctions permettant d'obtenir des informations issues du catalogue d'une base, ainsi que des informations sur un pilote ou une source de données

- Le niveau 2 (Level 2 API)

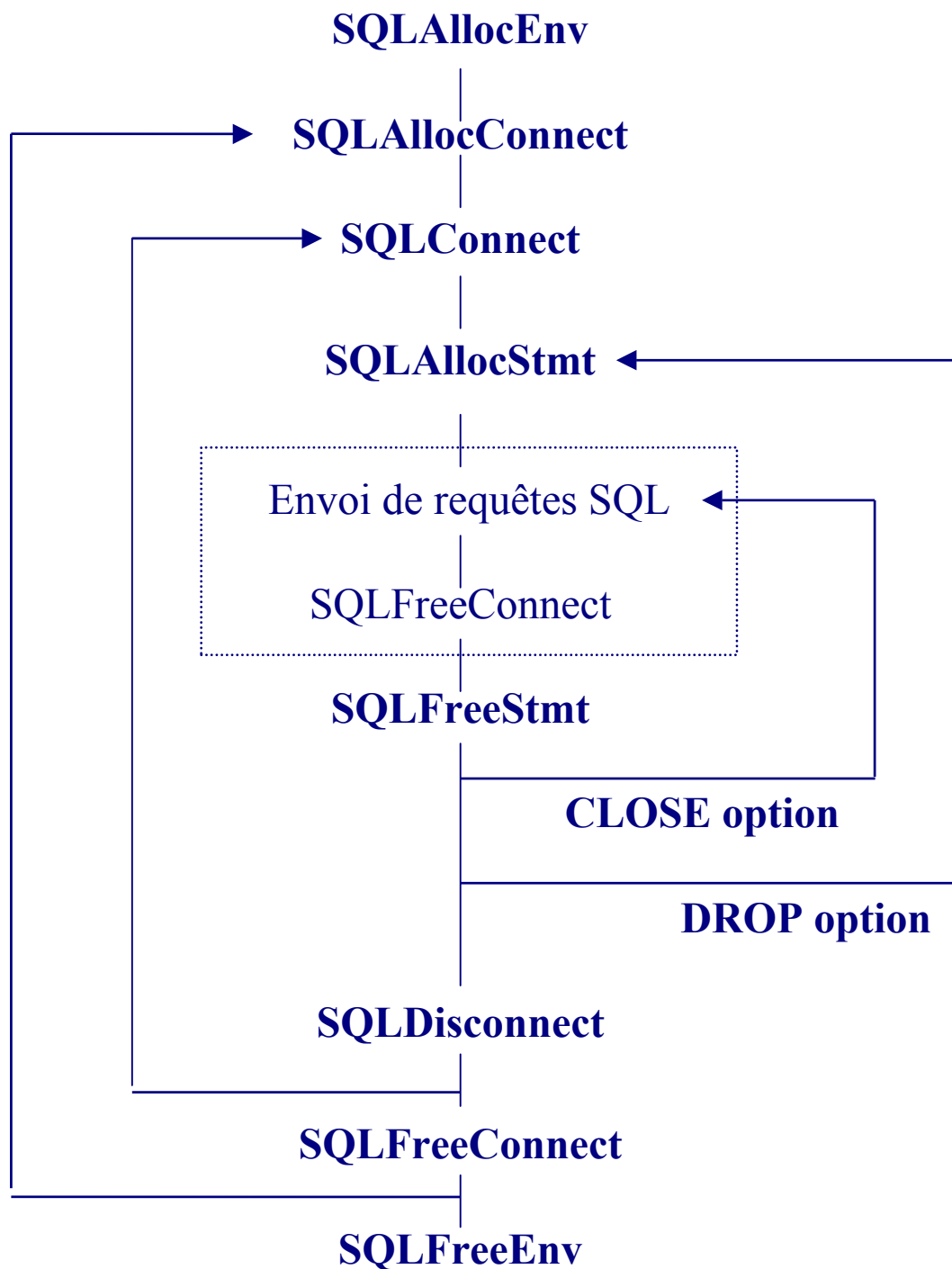
- ❖ Niveau 1 +

- ❖ Fonctions de gestion des curseurs

## Niveaux de conformité SQL

- Grammaire SQL minimale
  - LDD : CREATE et DROP TABLE
  - LMD : SELECT, INSERT, UPDATE, DELETE
  - Expressions simples dans les critères
- Grammaire SQL noyau
  - SQL min +
  - LDD : ALTER TABLE, CREATE INDEX, CREATE VIEW, GRANT, REVOKE
  - LMD : SELECT complet
- Grammaire SQL étendue
  - SQL Noyau +
  - LMD : jointure externes, unions
  - Procédures stockées

## 4. Structure d'un programme ODBC



## Structure d'un programme ODBC



## 5. Fonctions de connexion et de déconnexion

- SQLAllocEnv

- ❖ définit un descripteur d'environnement pour l'application
- ❖ ce descripteur est l'adresse d'une zone mémoire où seront placées des informations globales pour l'application, par exemple, le descripteur de la connexion courante

- SQLAllocConnect

- ❖ définit un descripteur de connexion
- ❖ ce descripteur est l'adresse d'une zone mémoire où seront placées des informations concernant une connexion
- ❖ un descripteur de connexion est toujours associé à un descripteur d'environnement

- **SQLConnect**

- ❖ charge un pilote et établit une connexion entre l'application et une source de données

- **SQLDisconnect**

- ❖ termine une connexion entre l'application et une source de données

- **SQLFreeConnect**

- ❖ libère un descripteur de connexion

- **SQLFreeEnv**

- ❖ libère un descripteur d'environnement

# Programmation

```
#include <windows.h>
#include <sql.h>
#include <sqlext.h>

HENV henv; // descripteur d'environnement
HDBC hdbc; // descripteur de connexion

// Allouer un descripteur d'environnement
SQLAllocEnv(&henv);

// Allouer un descripteur de connexion
SQLAllocConnect(henv, &hdbc);

// Etablir la connexion
SQLConnect( hdbc, "oracle", SQL_NTS,
            "scott", SQL_NTS,
            "tiger", SQL_NTS );

/* TRAITER LES REQUETES SQL */

// Terminer la connexion
SQLDisconnect(hdbc);

// Libérer les descripteurs
SQLFreeConnect(hdbc);
SQLFreeEnv(henv);
```

## 6. Transactions dans ODBC

Deux modes de validation des instructions SQL :

- Mode `AUTO_COMMIT`
  - chaque instruction SQL est automatiquement validée après son exécution
  - pas de notion de transaction dans ce mode
  - option par défaut
  - les pilotes qui ne supportent pas la notion de transaction sont toujours en mode `AUTO_COMMIT`
- Mode transactionnel
  - le programmeur gère explicitement la fin (validation ou annulation) des transactions

- **SQLConnectOptions**

- ❖ permet de spécifier différentes options de connexion, en particulier le mode de validation
- ❖ il faut utiliser SQLConnectOptions avant d'établir la connexion

- **SQLTransact**

- ❖ termine une transaction
  - soit en la validant \verb+SQL\_COMMIT+
  - soit en l'annulant \verb+SQL\_ROLLBACK+

# Programmation

```
SQLAllocEnv(&henv);

SQLAllocConnect(henv, &hdbc);

SQLSetConnectOption(hdbc,
    SQL_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF);

SQLConnect(hdbc, "oracle", SQL_NTS,
    "scott", SQL_NTS, "tiger", SQL_NTS );

/* mise a jour no 1 */
/* mise a jour no 2 */
/* mise a jour no 3 */

SQLTransact(henv, hdbc, SQL_COMMIT);

SQLDisconnect(hdbc);

SQLFreeConnect(hdbc);

SQLFreeEnv(henv);
```

L'appel à

```
SQLTransact(henv, hdbc, SQL_COMMIT)
```

permet de valider en bloc les 3 mises à jour effectuées dans le contexte de la connexion `hdbc`.

## 7. Envoi de requêtes SQL

Deux manières pour soumettre une requête SQL:

- Envoi pour exécution directe

- ❖ ce cas concerne les instructions qui ne seront exécutées qu'une seule fois

- ❖ l'instruction est préparée et exécutée en une seule étape au moyen d'un appel à `SQLExecDirect`

- Envoi pour préparation puis demandes d'exécution

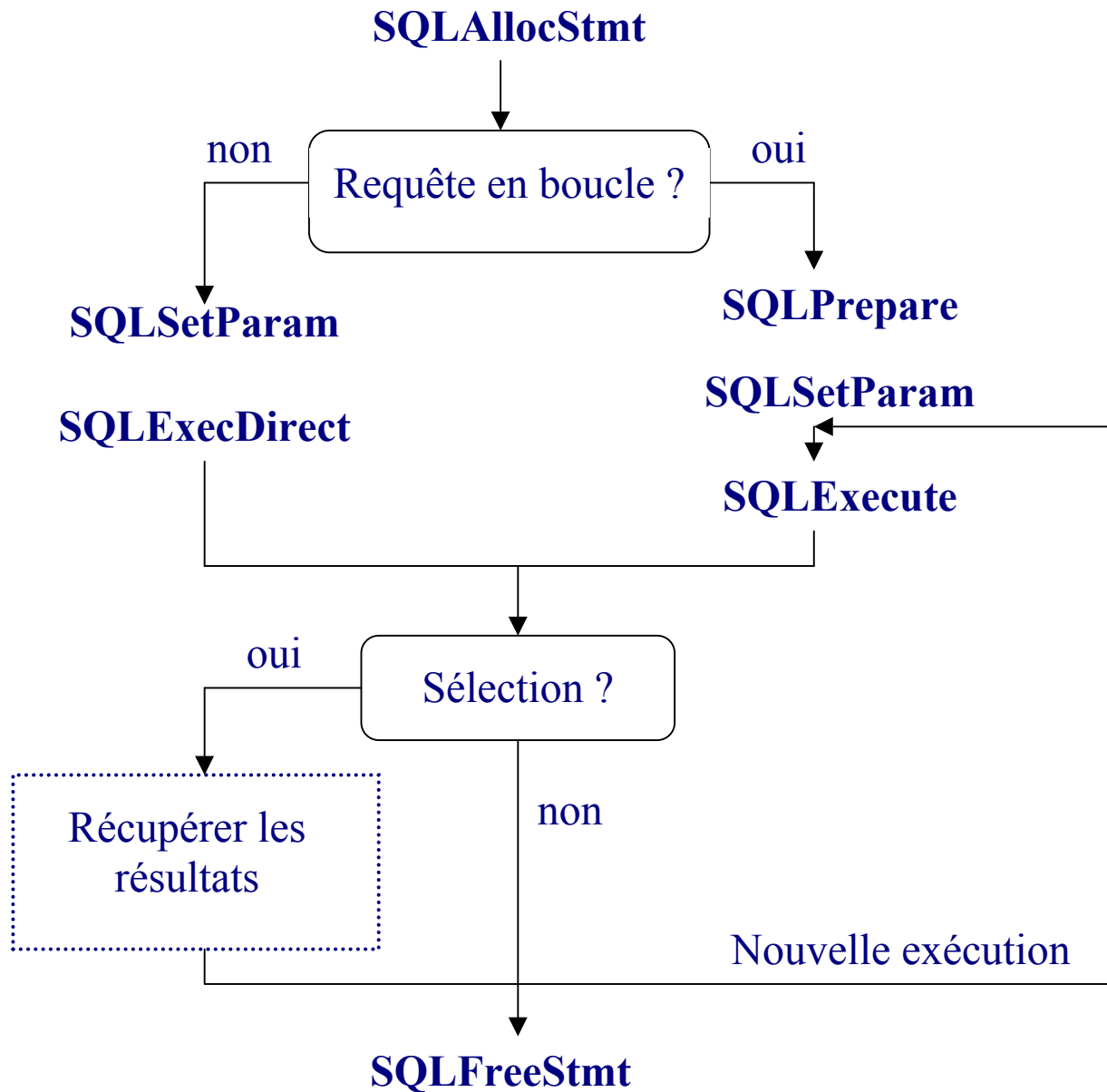
- ❖ ce cas concerne les instructions qui seront exécutées plusieurs fois

- ❖ l'instruction est préparée une seule fois en faisant appel à `SQLPrepare`

- ❖ l'instruction est ensuite exécutée au moyen de `SQLExecute`

- ❖ rien n'empêche d'exécuter plusieurs fois la même instruction en utilisant `SQLExecDirect`, mais c'est moins efficace

# Gestion d'une requête SQL



# Gestion d'une requête SQL



- Paramètres d'une requête SQL

- ❖ Les paramètres d'une requête peuvent être associés à des variables du programme, en utilisant la fonction `SQLSetParam`

- Fonctions de gestion des requêtes SQL

### **SQLAllocStmt**

- ❖ définit un descripteur d'instruction et l'associe à une connexion

### **SQLFreeStmt**

- ❖ libère un descripteur d'instruction
- ❖ si un curseur est associé à l'instruction, il est fermé

### **SQLExecDirect**

- ❖ prépare et exécute directement une instruction SQL

## **SQLPrepare**

- ❖ prépare une instruction SQL

## **SQLExecute**

- ❖ exécute une instruction SQL préparée en utilisant les valeurs courantes des éventuels paramètres
- ❖ on utilise cette fonction lorsqu'on doit exécuter plusieurs fois la même instruction dans l'application, dans ce cas, l'instruction n'est préparée qu'une seule fois

## **SQLSetParam**

- ❖ permet d'associer à un paramètre d'une instruction SQL, une variable contenant la valeur du paramètre
- ❖ l'utilisation de cette fonction est déconseillé depuis ODBC v2.0 où elle a été remplacée par SQLBindParameter mais qui est une fonction du niveau 1

- Terminer le traitement d'une instruction SQL
  - ❖ La fonction **SQLFreeStmt** permet de libérer les ressources associées à un descripteur d'instruction
  - ❖ Elle possède quatre options:

### **SQL\_CLOSE**

- Ferme le curseur éventuellement
- Le descripteur d'instruction peut être utilisé à nouveau

### **SQL\_DROP**

- Ferme le curseur éventuellement
- Libère toutes les ressources associées au descripteur d'instruction

### **SQL\_UNBIND**

- Libère tous les buffers liés par **SQLBindCol**

### **SQL\_RESET\_PARAMS**

- Libère tous les buffers requis par **SQLBindParameter**

# Programmation

```
rc = SQLAllocEnv(&henv) ;

rc = SQLAllocConnect(henv, &hdbc) ;

rc = SQLConnect(hdbc, "oracle", SQL_NTS,
               "scott", SQL_NTS, "tiger", SQL_NTS) ;

rc = SQLAllocStmt(hdbc, &hstmt) ;

rc = SQLExecDirect(hstmt,
                  "select * from employe", SQL_NTS) ;


/* RECUPERATION DES RESULTATS */


SQLFreeStmt(hstmt, SQL_UNBIND) ;

SQLFreeStmt(hstmt, SQL_DROP) ;

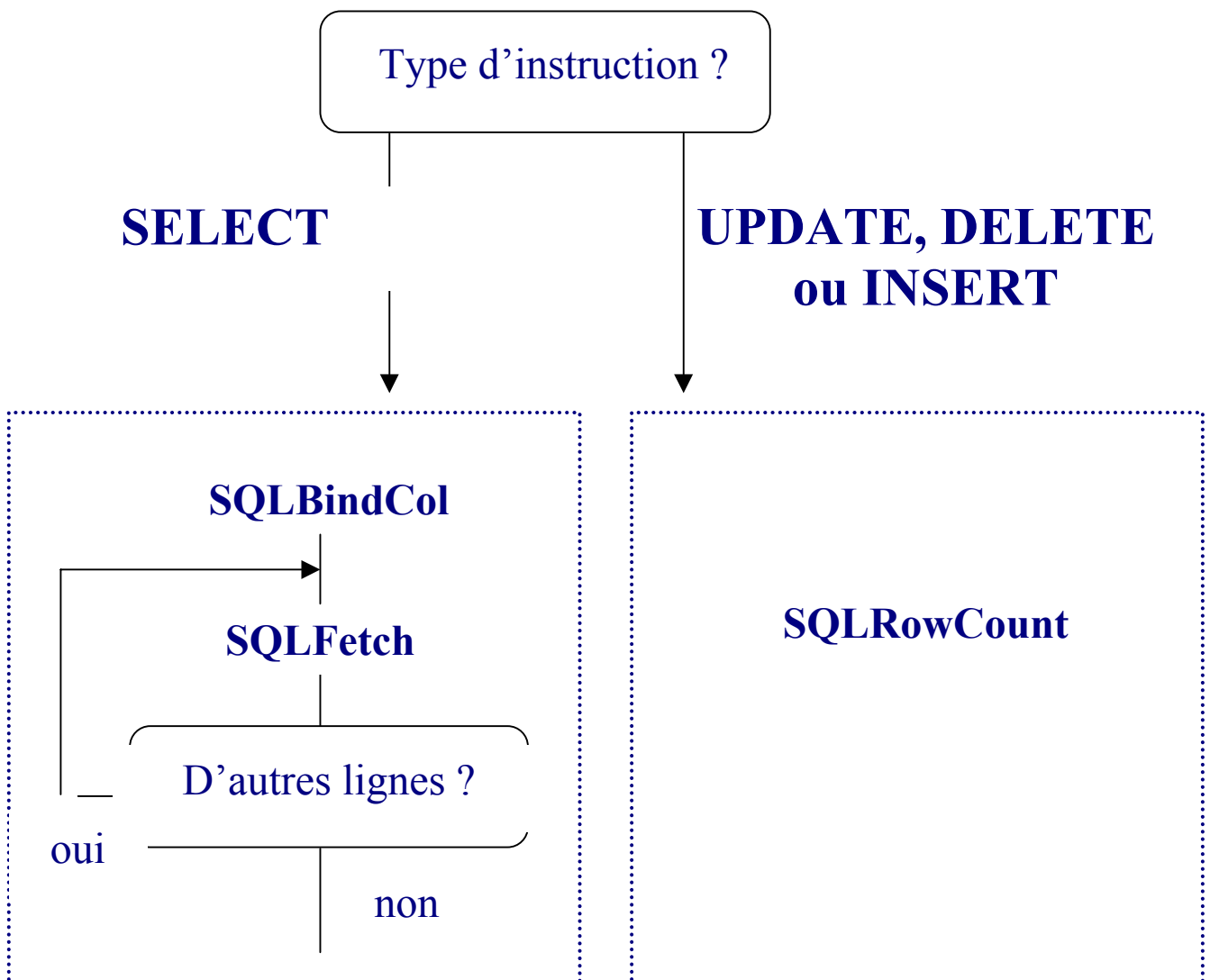
SQLFreeStmt(hstmt, SQL_CLOSE) ;

SQLDisconnect(hdbc) ;

SQLFreeConnect(hdbc) ;

SQLFreeEnv(henv) ;
```

## 8. Récupération des résultats



Récupération des résultats

- Liaison d'une colonne à une zone mémoire (BINDING)
  - ❖ L'association d'une zone mémoire à une colonne de l'ensemble résultat se fait en utilisant la fonction **SQLBindCol**
  - ❖ Paramètres de **SQLBindCol**
    - un descripteur d'instruction
    - le numéro de la colonne résultat
    - le type C de la colonne
    - l'adresse de la variable qui recevra les valeurs de cette colonne
    - le nombre d'octets maximum de la zone mémoire
    - le nombre d'octets écrits dans la zone mémoire
- Récupérer les lignes (FETCH)
  - ❖ Les lignes de l'ensemble résultat sont récupérées en utilisant la fonction **SQLFetch**

# Programmation

```
rc = SQLExecDirect(hstmt,
    "select no, nom from employe", SQL_NTS);

rc = SQLBindCol(hstmt, 1,
    SQL_C_FLOAT, &no, 0, &cbno);

rc = SQLBindCol(hstmt, 2,
    SQL_C_CHAR, &nom, 20+1, &cbnom);

while (1) {

    rc = SQLFetch(hstmt);

    if (rc == SQL_NO_DATA_FOUND) break;

    if (rc != SQL_SUCCESS) {
        printf("\n**Erreur fatale...\n");
        break;
    }

    printf("%f %20s", no, nom);

}

SQLFreeStmt(hstmt, SQL_UNBIND);

SQLFreeStmt(hstmt, SQL_CLOSE);
```

## 9. Détection des erreurs

- Codes de retour des fonctions
  - ❖ **SQL\_SUCCESS**
  - ❖ **SQL\_SUCCESS\_WITH\_INFO**
  - ❖ **SQL\_NO\_DATA\_FOUND** aucune ligne retournée avec **FETCH**
  - ❖ **SQL\_ERROR**
  - ❖ **SQL\_INVALID\_HANDLE**
  - ❖ **SQL\_STILL\_EXECUTING**
  - ❖ **SQL\_NEED\_DATA**
- Récupérer les messages d'erreurs
  - ❖ La fonction **SQLError** permet d'obtenir des informations supplémentaires, lorsqu'une fonction ODBC retourne le code **SQL\_ERROR** ou **SQL\_SUCCESS\_WITH\_INFO**