Nicolas Loménie

Bases de Données

Plan du Cours d'Initiation

3. Qu'est-ce qu'une BD ? Notion de SGBDR.
4. Modéliser conceptuellement la BD
5. Modéliser mathématiquement la BD
6. Comprendre l'Algèbre Relationnelle
7. Se familiariser avec SQL, le langage informatique
8. Aller plus loin avec SQL et l'Algèbre Relationnelle
9. Appréhender la notion de transactions et de vues
10. Comprendre les subtilités d'implémentation physique

Bases de Données

Nicolas Loménie

INTRODUCTION

INTRODUCTION

Qu’est-ce qu’une BD ? Notion de SGBDR.

Dupont

Symptomes : y
Turlututu : sqj
Symptomes : y
Turlututu : sdd
Analyses : xxx

Dupond

Turlututusqjsk
Symptom: yyyy
Analyses xxxx

Turlututudhjsd
Analyses :xx

L'APPROCHE SYSTEMES DE FICHIERS

Duhpon

Symptomes : yy
Analyses : xxxx

Symptomes : yy

Duipont

Turlututu : sq

Symptomyyyy
Analysesxxxx

Turlututudhjsd

Caractéristiques

Plusieurs applications

- ➔ plusieurs formats
- ➔ plusieurs langages

Redondance de données

Pas de facilité d'interrogation

- ➔ Question ⇒développement

Redondance de code

Problèmes

- ➔ **Difficultés de gestion**
- ➔ **Incohérence des données**
- ➔ **Coûts élevés**
- ➔ **Maintenance difficile**
- ➔ **Gestion de pannes ???**
- ➔ **Partage des données ???**
- ➔ **Confidentialité ???**

L'approche ''Bases de données''

- **Modélisation des données**
  - ➔ Eliminer la **redondance** de données
  - ➔ **Centraliser** et **organiser** correctement les données
  - ➔ Plusieurs niveaux de modélisation
  - ➔ Outils de conception

- **Logiciel «Système de Gestion de Bases de Données»**
  - ➔**Factorisation** des modules de contrôle des applications
    - Interrogation, cohérence, partage, gestion de pannes, etc…
  - ➔Administration facilitées des données

Système de gestion de bases de données

I- Indépendance Physique

II- Indépendance Logique

III – Langage de manipulation

IV - Gestion des vues

V - Optimisation des questions

VI - Gestion de la cohérence

VII - Gestion des pannes

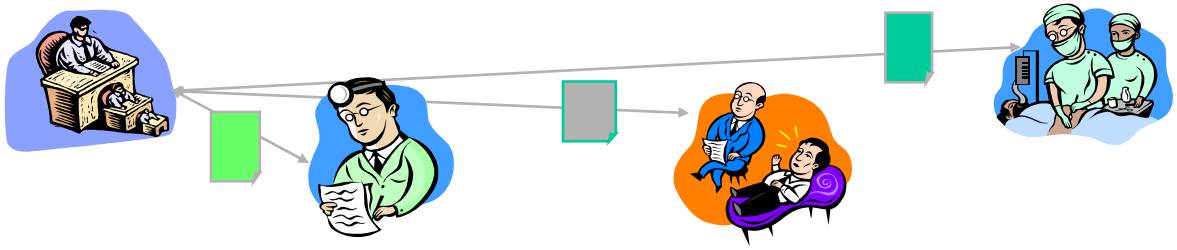
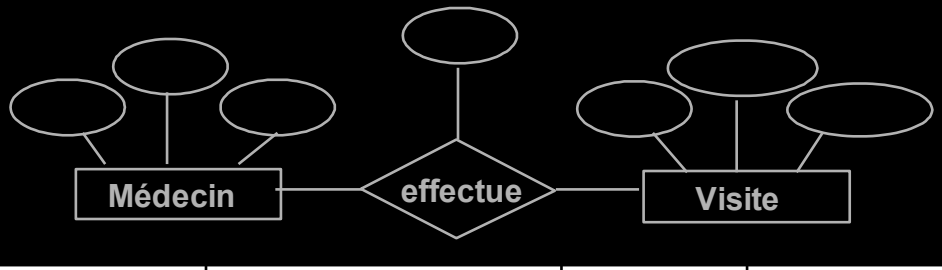
VIII - Concurrence d'accès

IX - Gestion de la confidentialité

X - Standards

BD

Modélisation du réel

Réel					
Modèle conceptuel	• Indépendant du modèle de données • Indépendant du SGBD				
Modèle logique	• Dépendant du modèle de données • Indépendant du SGBD	**Codasyl**	**Relationnel**	**Objet**	**XML**
Modèle Physique	• Dépendant du modèle de données • Dépendant du SGBD	• **Organisation physique des données** • **Structures de stockage des données** • **Structures accélératrices (index)**			

IMPLEMENTATION

- La manipulation se fait via un langage **déclaratif**
  - La question déclare l'objectif sans décrire la méthode
  - Le langage suit une norme commune à tous les SGBD
  - **SQL : Structured Query Langage**

- Syntaxe (aperçu !)

Select **Nom, Description**
From **Médicaments**
Where **Type = 'Aspirine'**

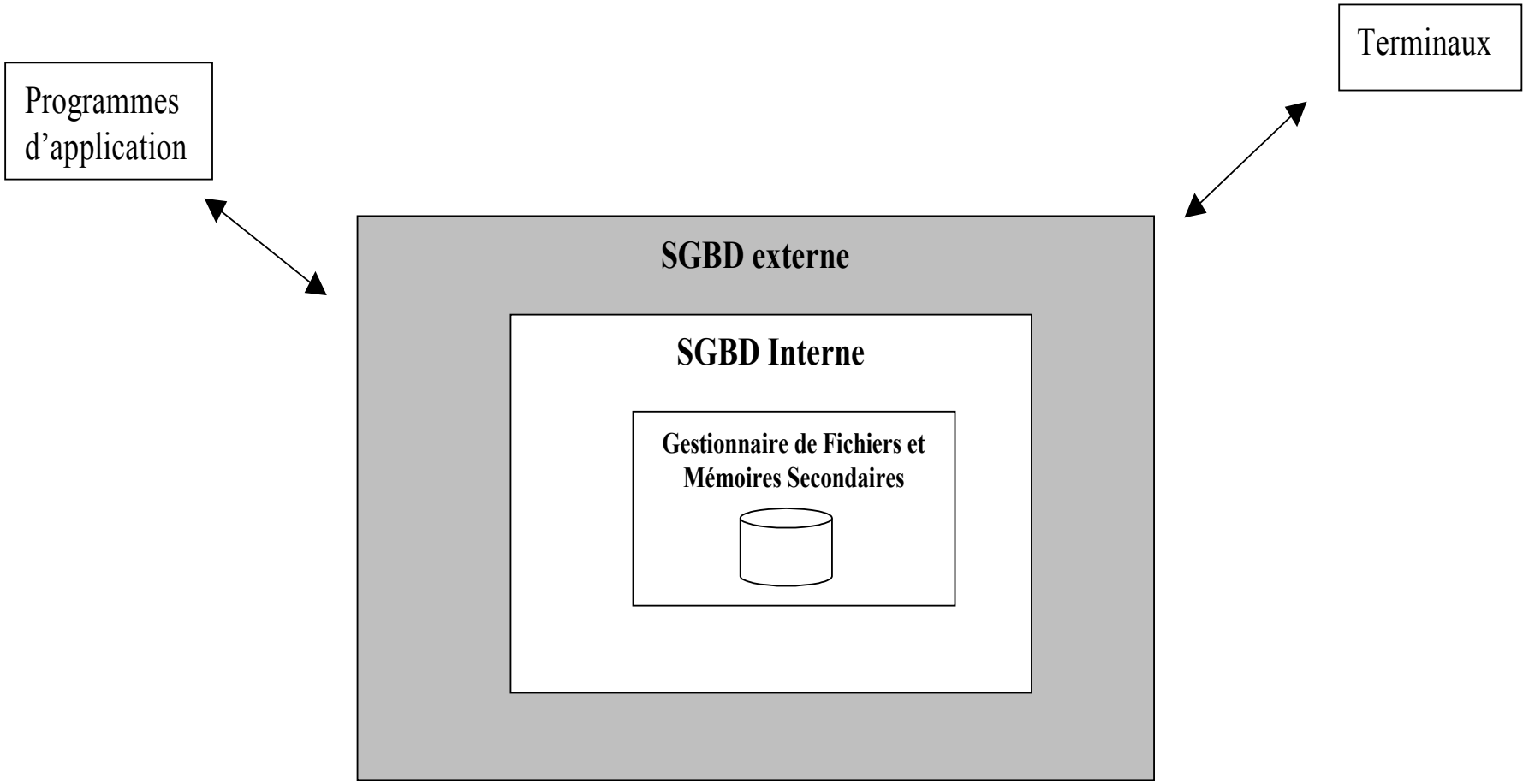
- 1970 : bases théoriques au modèle relationnel : Algèbre de Codd

- 1980 : Logiciel de SGBDR commercialisés : ORACLE, SYBASE, SQL SERVER et Access

- Logiciels libres : mysql; postgresl

- Actuellement, en germe, SGBDO (Oracle 8 : SGBDOR), Multimédia, Data Mining …

Structuration en trois couches :



Description des données en trois niveaux d'abstraction

Schéma conceptuel :

description des données d'une organisation en terme de types d'objets et de liens logiques indépendants de toute représentation en machine, correspondant à une vue canonique globale de l'organisation modélisée
-> Schéma ER
-> Administrateur Organisation

Schéma interne :

description des données d'une base en termes de représentation physique en machine, correspondant à une spécification des structures de mémorisation et des méthodes de stockage et d'accès utilisées pour ranger et retrouver les données sur disque
-> INDEX
-> Administrateur BD

Schéma externe :

description d'une partie de la base de données, extraite ou calculée à partir de la base physique, correspondant à la vision d'un programme ou d'un utilisateur, donc à un arrangement particulier de certaines données
-> VUES
-> Administrateur Applications

5 étapes pour l'élaboration des schémas conceptuels et internes (coeur de la BDR)

1. Perception du monde réel et capture des besoins
   Entretiens -> ensemble de vues ou schémas externes

3. Elaboration du schéma conceptuel
   ensemble de vues ou schémas externes -> intégration dans un schéma global

5. Conception du schéma logique (automatisable)
   schéma global -> ensemble de tables

7. Affinement du schéma logique
   ensemble de tables -> ensemble de tables regroupées ou décomposées

9. Elaboration du schéma physique
   ->ensemble de tables regroupées ou décomposées indexées

Bases de Données

Nicolas Loménie

MODELE CONCEPTUEL

MODELE CONCEPTUEL

Modéliser conceptuellement la BD

Objectifs :

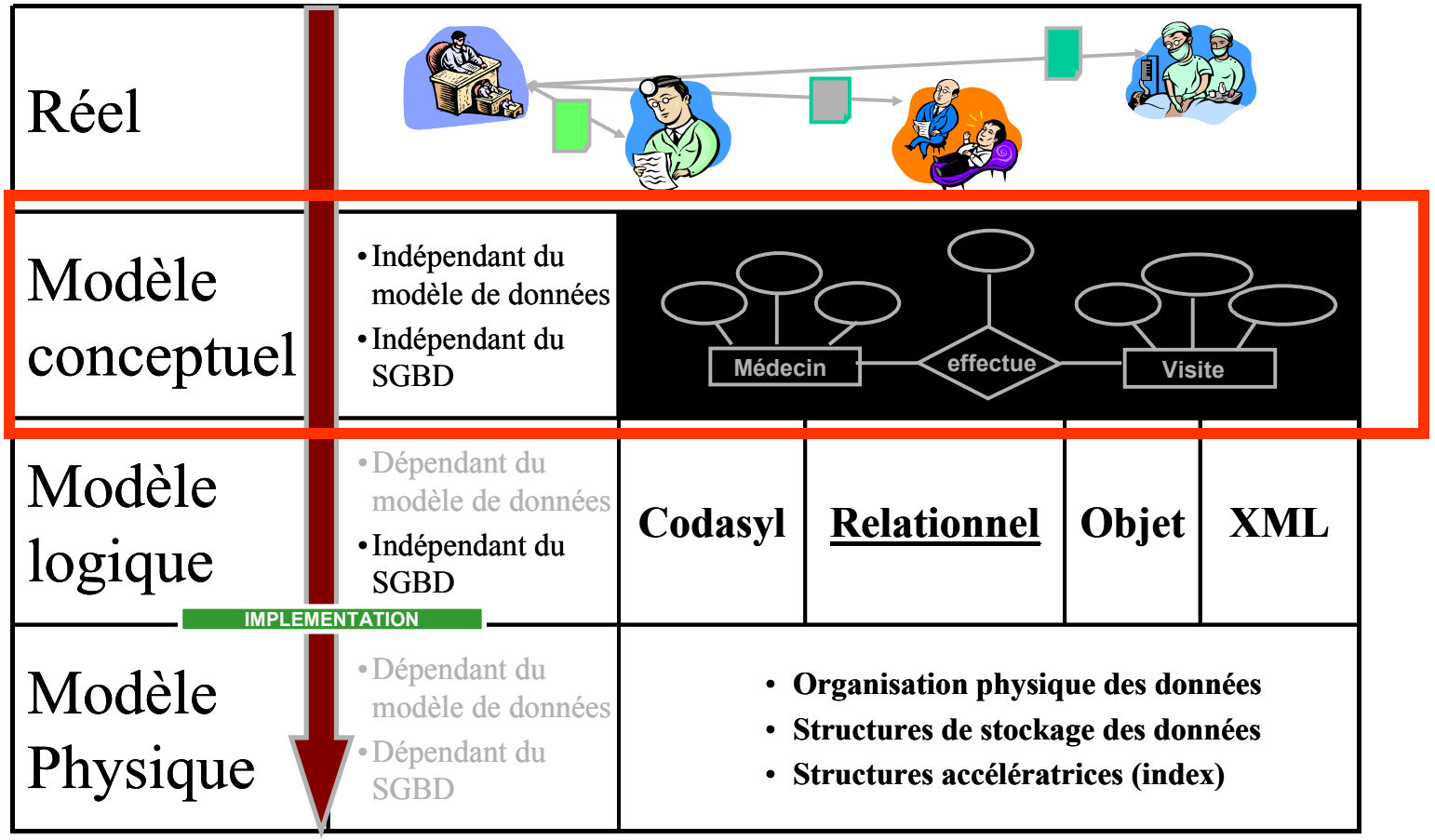
- Prendre en main la méthodologie de conception MERISE

- Apprendre les définitions essentielles des objets manipulés par le modèle conceptuel : entité, attribut, identifiant, relation, dimension, cardinalité.

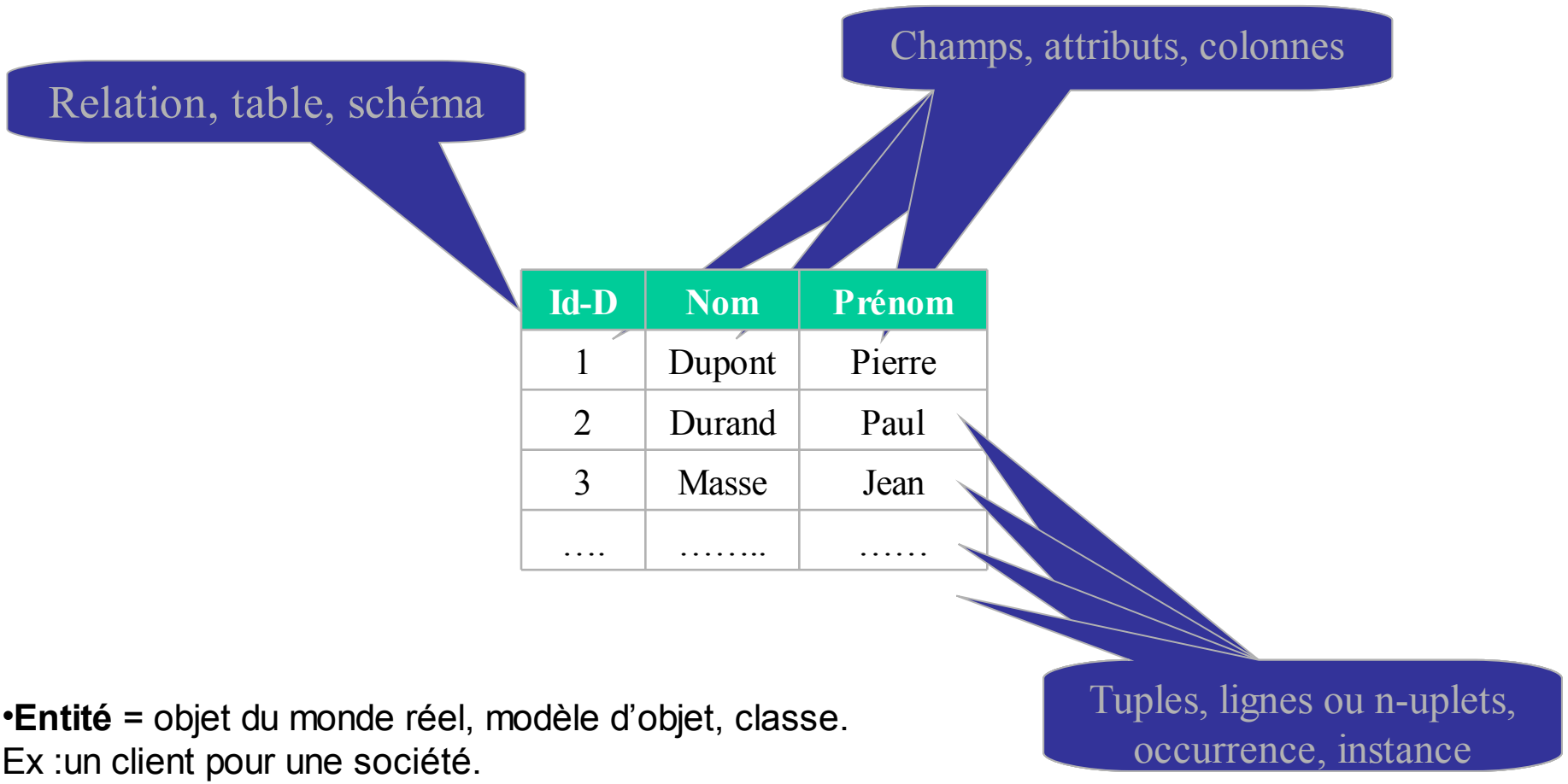
- Etre capable d'appréhender la méthodologie de passage d'un énoncé en langage naturel à un modèle conceptuel, avec la part d'arbitraire inhérente.

- Plusieurs noms dans le cadre d'un SGBDR :
  Modèles Entités-Associations, Schéma ER, MCD

- Nécessité d'une méthodologie et d'un cadre de conception : MERISE, UML …



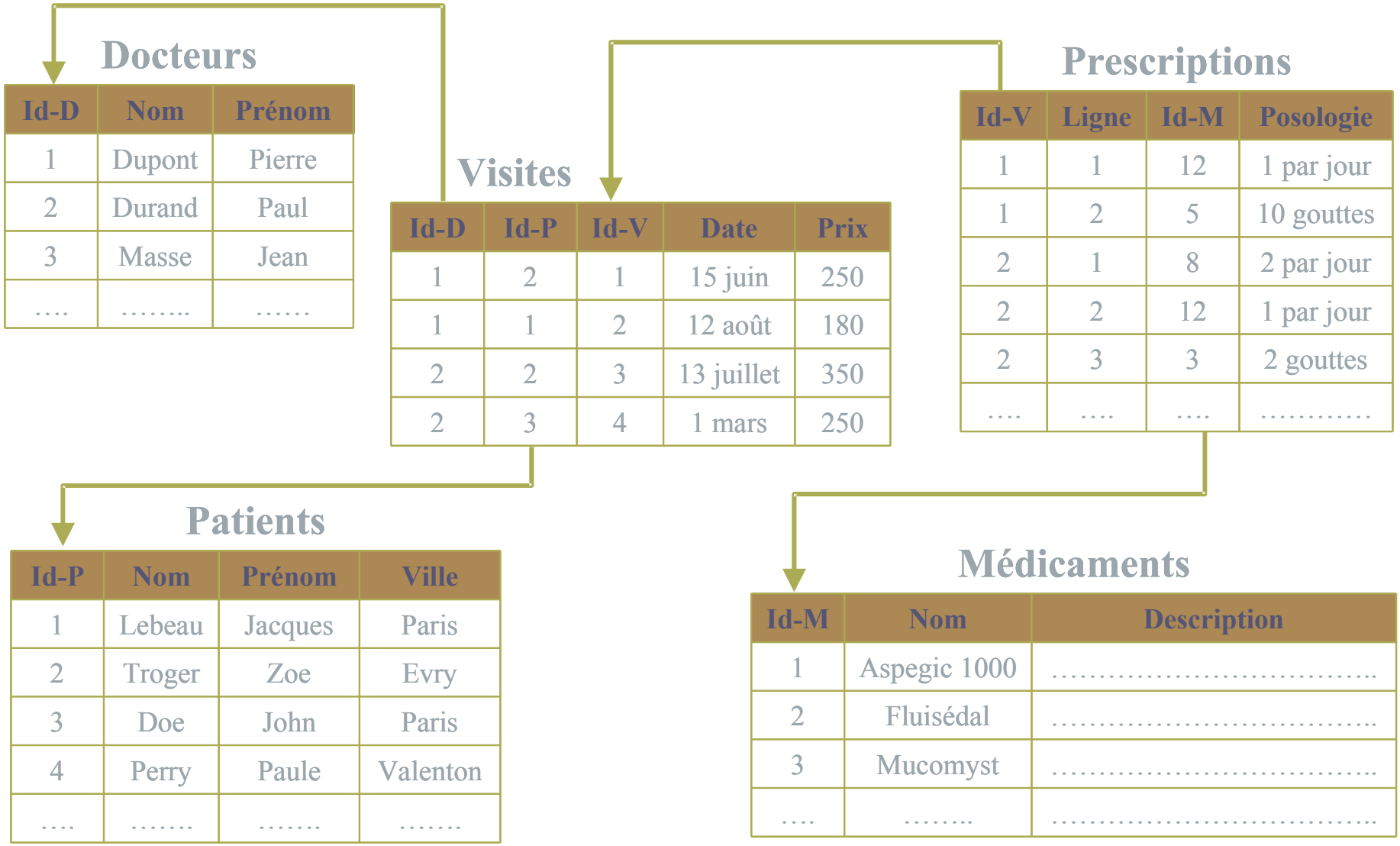
• Le modèle de données dit Entités-Relations (MERISE) ou Entités-Associations



•**Entité** = objet du monde réel, modèle d'objet, classe.
Ex :un client pour une société.

•**Attributs** (caractéristiques ou propriétés) = ils décrivent les entités ;

•**Identifiant** = ensemble d'attributs permettant de déterminer une et une seule occurence d'une entité ;



Docteurs

Id-D	Nom	Prénom
1	Dupont	Pierre
2	Durand	Paul
3	Masse	Jean
….	……..	……

Visites

Id-D	Id-P	Id-V	Date	Prix
1	2	1	15 juin	250
1	1	2	12 août	180
2	2	3	13 juillet	350
2	3	4	1 mars	250

Prescriptions

Id-V	Ligne	Id-M	Posologie
1	1	12	1 par jour
1	2	5	10 gouttes
2	1	8	2 par jour
2	2	12	1 par jour
2	3	3	2 gouttes
….	….	….	……

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton
….	…….	…….	…….

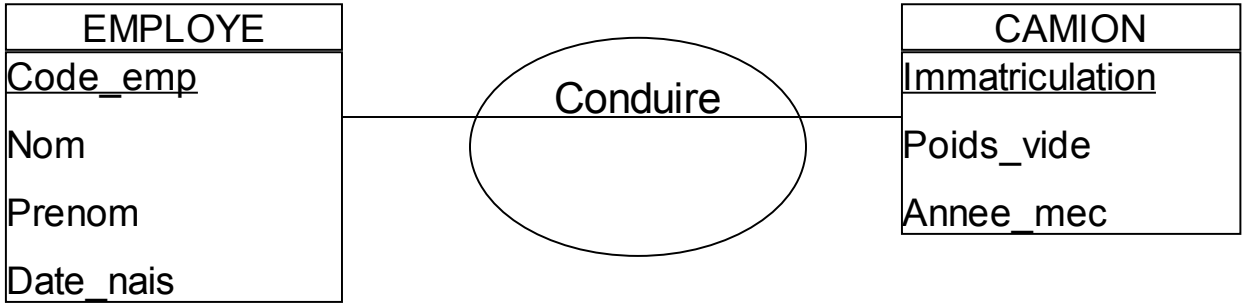
Médicaments

Id-M	Nom	Description
1	Aspegic 1000	……..
2	Fluisédal	……..
3	Mucomyst	……..
….	……..	……..

•**Relation** = représente les liens entre les entités ; contrairement aux entités, les relations n'ont pas d'existence propre ; mais comme les entités, elles sont caractérisées par un nom et d'éventuels attributs.

Représentation graphique de ces schémas :

- entités par rectangles;
- relations par ellipses ;
- les attributs identifiants sont soulignés ;



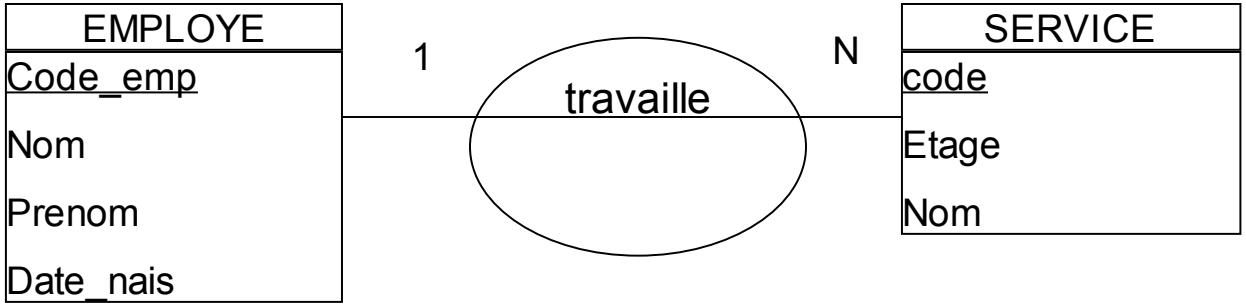
•**Dimension (ou degré) =** nombre d'entités impliquées dans la relation ;

•**Cardinalité** (maximale) **=** nombre de participations maximal d'une entité à une relation (une relation binaire peut être de cardinalité I-I, I-N ou M-N) ;

-> traduit les règles et contraintes propres à un processus métier

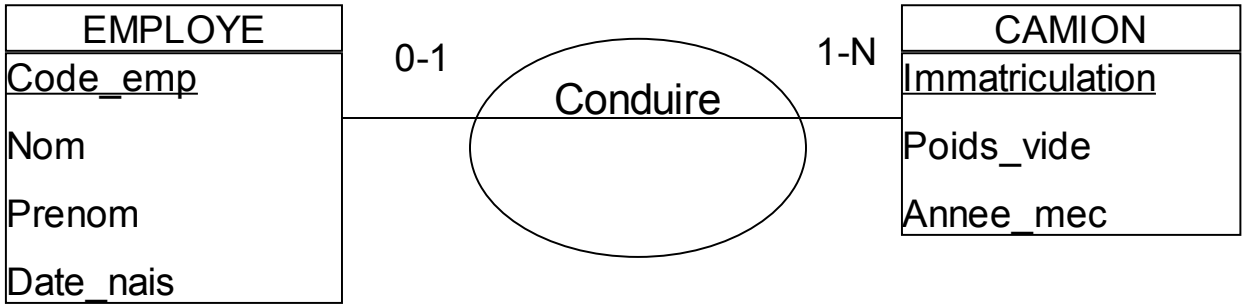
-> approche française (MERISE) : nombre de participations d'une entité à une relation

-> approche anglosaxone : nombre de correspondants d'une entité au sein d'une relation
Cette distinction a une importance pratique dans le cas de relation n-aire (avec n>2)

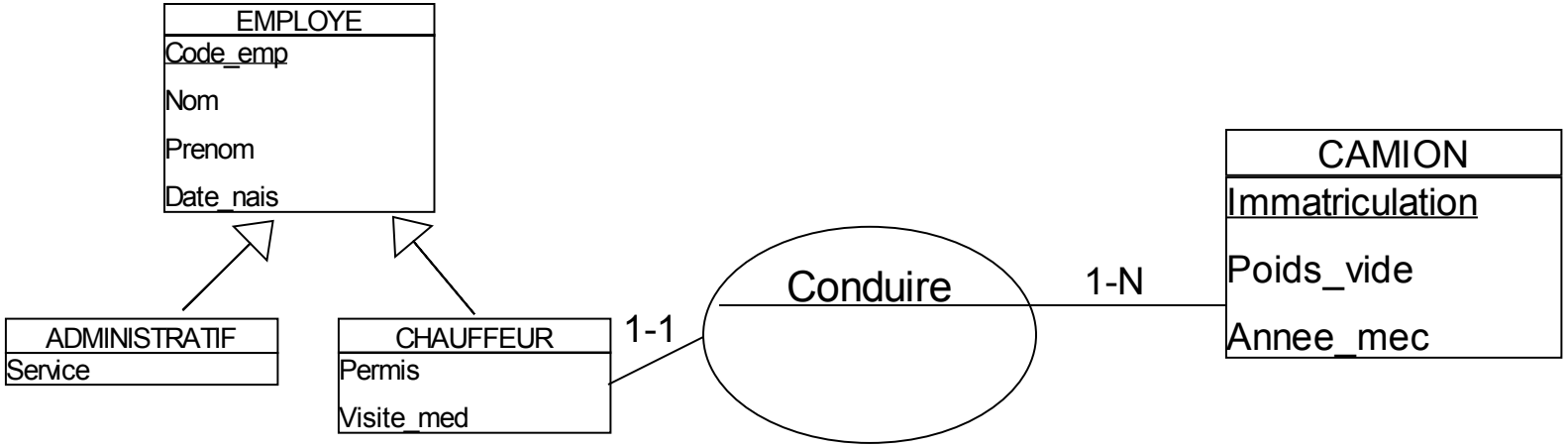


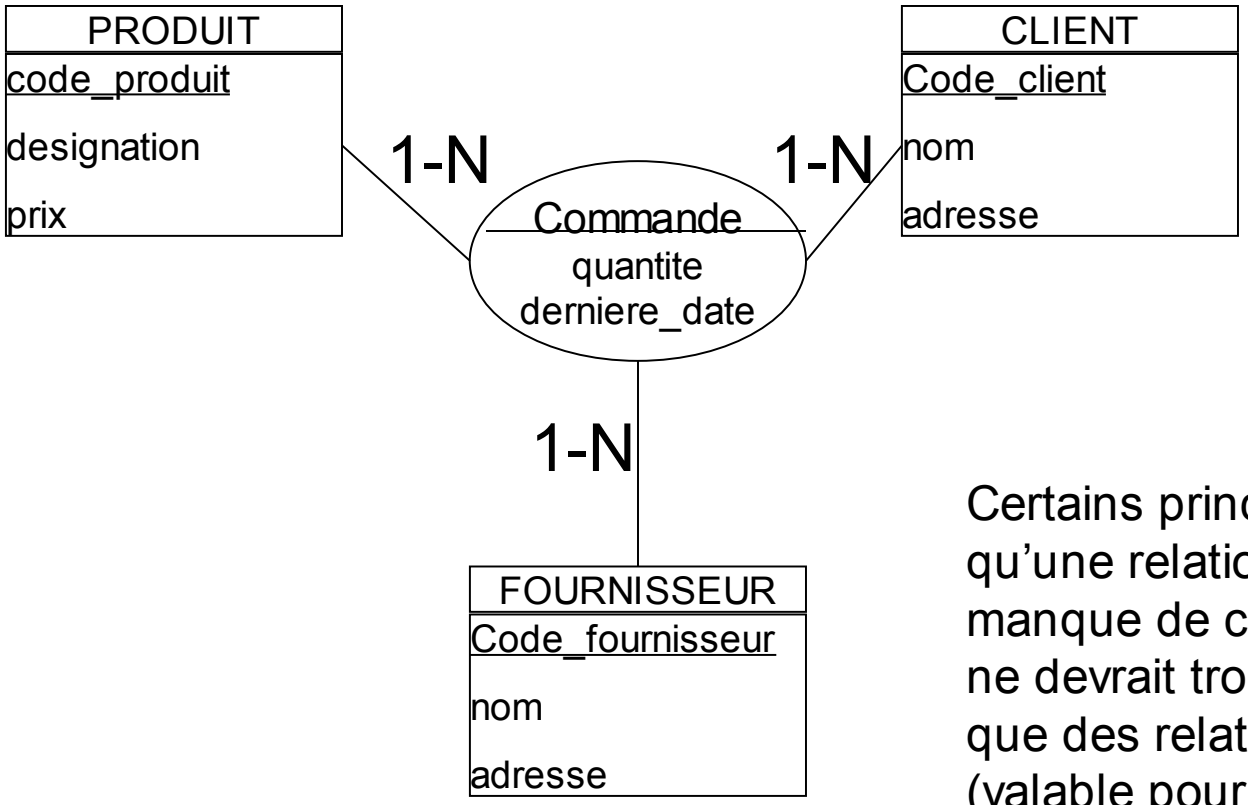
Un modèle EER enrichi :

- **Cardinalité minimale** ;



- **Généralisation** = description d'une entité à différents niveaux d'abstraction ;





Certains principes peuvent indiquer qu'une relation ternaire 1-N-N manque de cohérence et que l'on ne devrait trouver dans un MCD que des relations ternaires N-N-N (valable pour toutes relations n-aire avec n>2).

Ce principe permet de simplifier la suite du processus, et indique qu'une telle relation devrait pouvoir être découper en deux relations binaires par exemple.

La démarche pour passer de l'énoncé du problème au MCD:

- Déterminer la liste des entités ;

- Pour chaque entité :
  a) Etablir la liste de ses attributs ;
  b) Parmi ceux-ci, déterminer un identifiant ;

- Déterminer les relations entre les entités ;

- Pour chaque relation :
  a) Dresser la liste des attributs propres à la relation ;
  b) Vérifier la dimension (binaire, ternaire…) ;
  c) Définir les cardinalités (I-I, I-N, M-N) ;

- Vérifier le schéma obtenu, notamment :
  a) Supprimer les transitivités ;
  b) S'assurer que le schéma est connexe ;

• A toute situation à modéliser, 

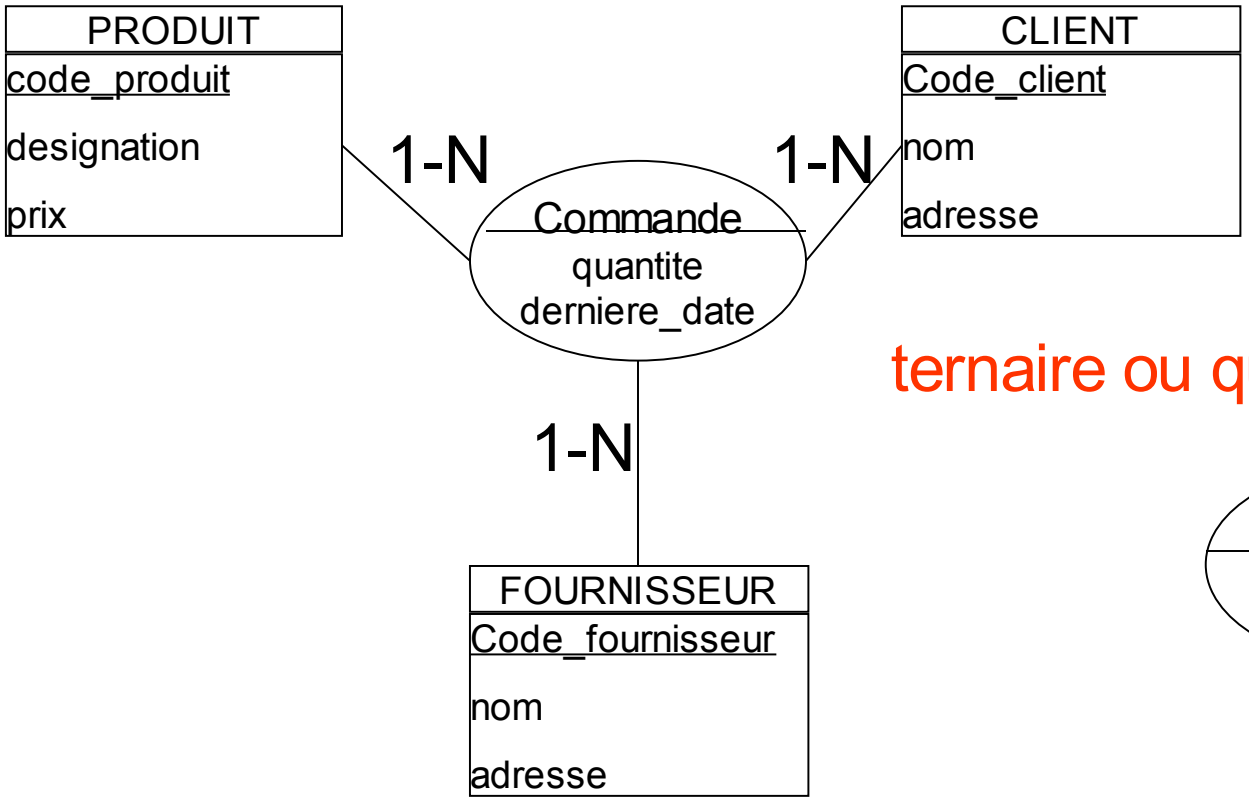
peuvent correspondre plusieurs schémas différents,

avec leurs avantages et leurs inconvénients.

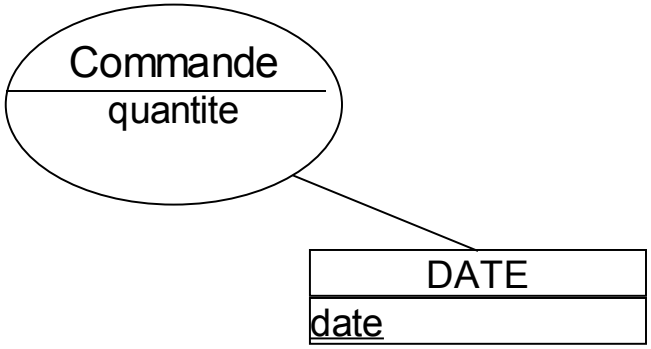
• Qualité d'une modélisation ER ou ERR :

l'expressivité,
la minimalité,
la lisibilité
et la simplicité.

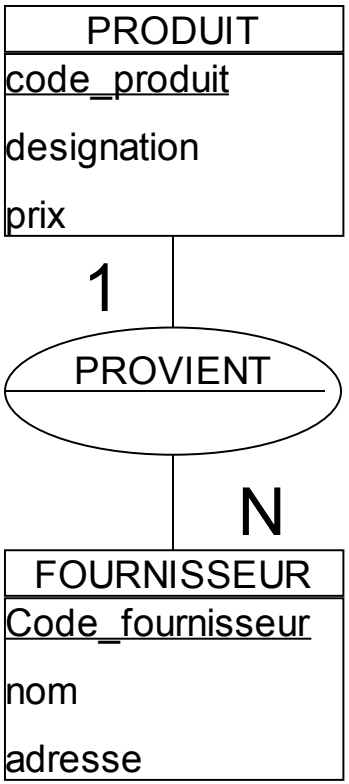
Ne pas surestimer la dimension d'une relation !

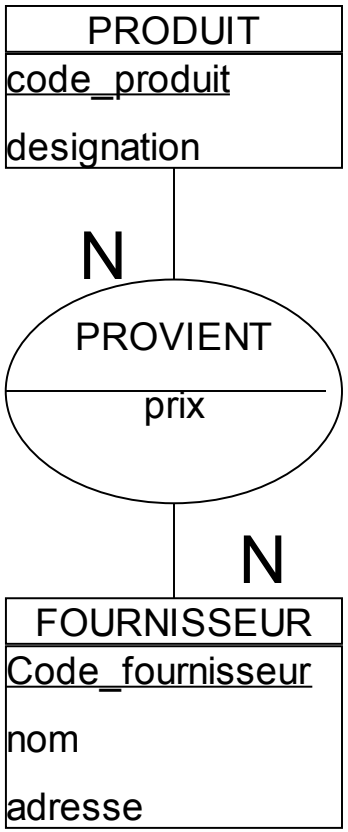


ternaire ou quaternaire ?

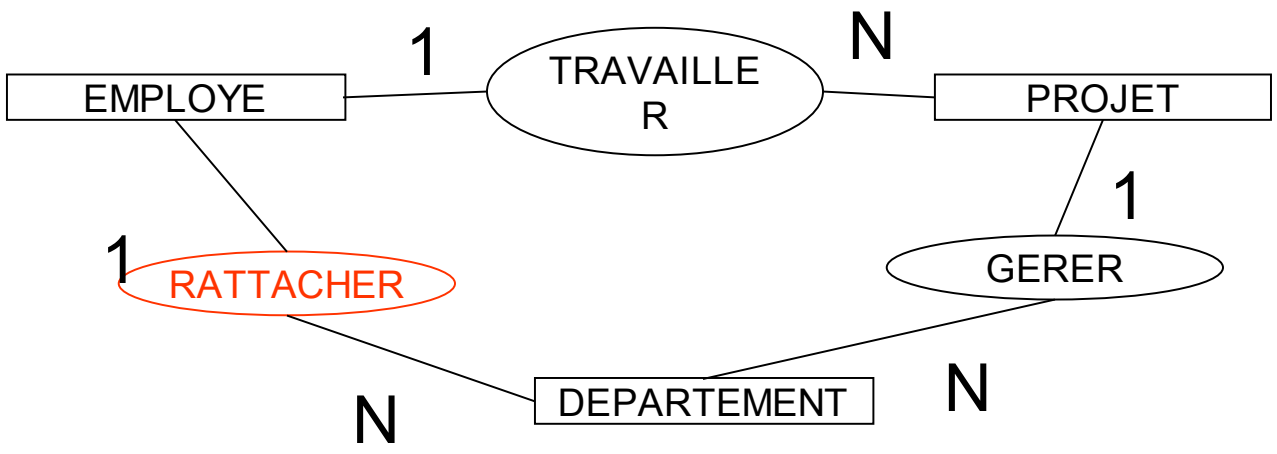


Ne pas attribuer à une relation les attributs des entités participantes ou inversement !





- Ne pas exprimer des relations redondantes !



- Ne pas se tromper de niveau de discours !

  -> Cas d'agences ou d'une agence

- Ne pas confondre les concepts de données et de traitement !

  -> Cas de Demander et Facturer

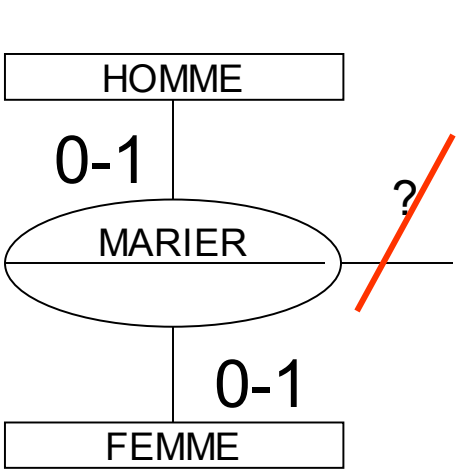
- Ne pas introduire d'attribut calculé !

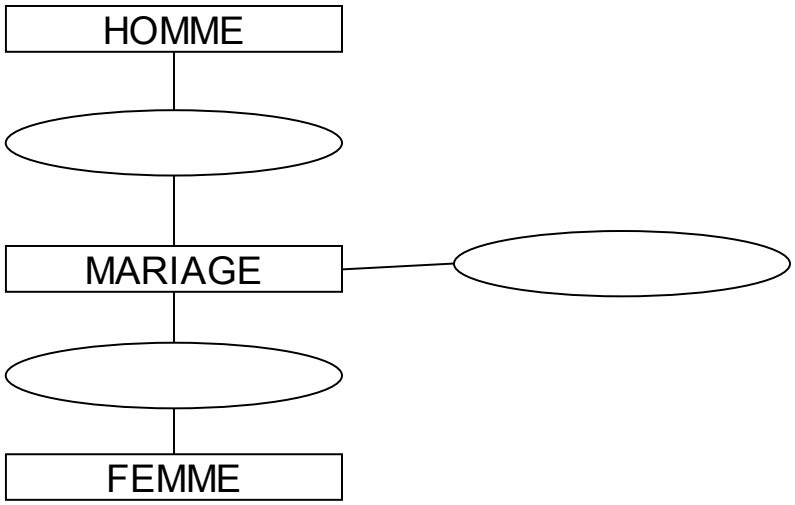
• Le conflit entité contre attribut

-> Cas du pays

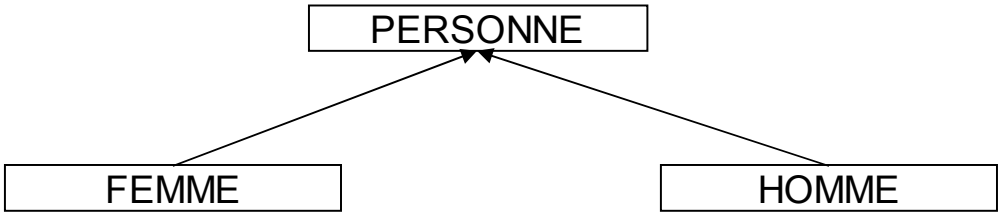
• Le conflit entité contre association

-> Cas de la commande, de la facture





• Le conflit généralisation contre attribut



Cas pratique : gestion des hôpitaux d'un département ;

Enoncé :

La direction départementale d'Ilde de France souhaite informatiser la gestion des hôpitaux du département. Chaque hôpital a des activités de soin dans les services médicaux et des activités de recherche dans les laboratoires. Les médecins employés sont obligatoirement rattachés à un seul hôpital. Ils ont le choix entre trois types de fonctions : consultant indépendant, médecin praticien, médecin chercheur. Les consultants indépendants ne sont rattachés à aucune structure interne de laboratoire ou de service. Les praticiens sont rattachés à un seul service. Les chercheurs sont rattachés à un laboratoire unique. Les fonctions de praticien et de chercheur peuvent être cumulées. Lorsqu'un médecin est consultant indépendant ou médecin praticien, il se voit confier la responsabilité d'un ou plusieurs patients. Dans certains cas nécessitant des traitements complexes, un même patient peut être suivi par plusieurs médecins.
Vous devez concevoir le schéma de la base de données d'Ile de France en tenant compte de ces hypothèses. Les hôpitaux, en plus de leurs nom et adresse, sont identifiés au moyen d'un code. De même, chaque service hospitalier et chaque laboratoire sont décrits par un code d'identification et un nom. On connaît, pour chaque médecin, son numéro matricule, ses coordonnées (nom, prénom, sexe, adresse, date de naissance) et sa spécialité (urologie, gynécologie, gériatrie, pédiatrie, etc.). Enfin, chaque patient possède un numéro de sécurité sociale, un nom, un prénom, une adresse, une date de naissance.

Les fonctions que le système devra effectuer sont :

- La gestion d'activité des médecins ;
- La gestion du suivi des patients ;
- Les statistiques départementales (nombre de laboratoires par domaine, nombre de médecins, etc.).

Construire le schéma ER modélisant cette situation, le justifier en fonction de vos hypothèses et indiquer le contenu des entités et relations. Constater les insuffisances du modèle ER pour représenter ces informations. Proposer plusieurs modélisations EER de cette situation et les justifier. Analyser leurs qualités comparées.

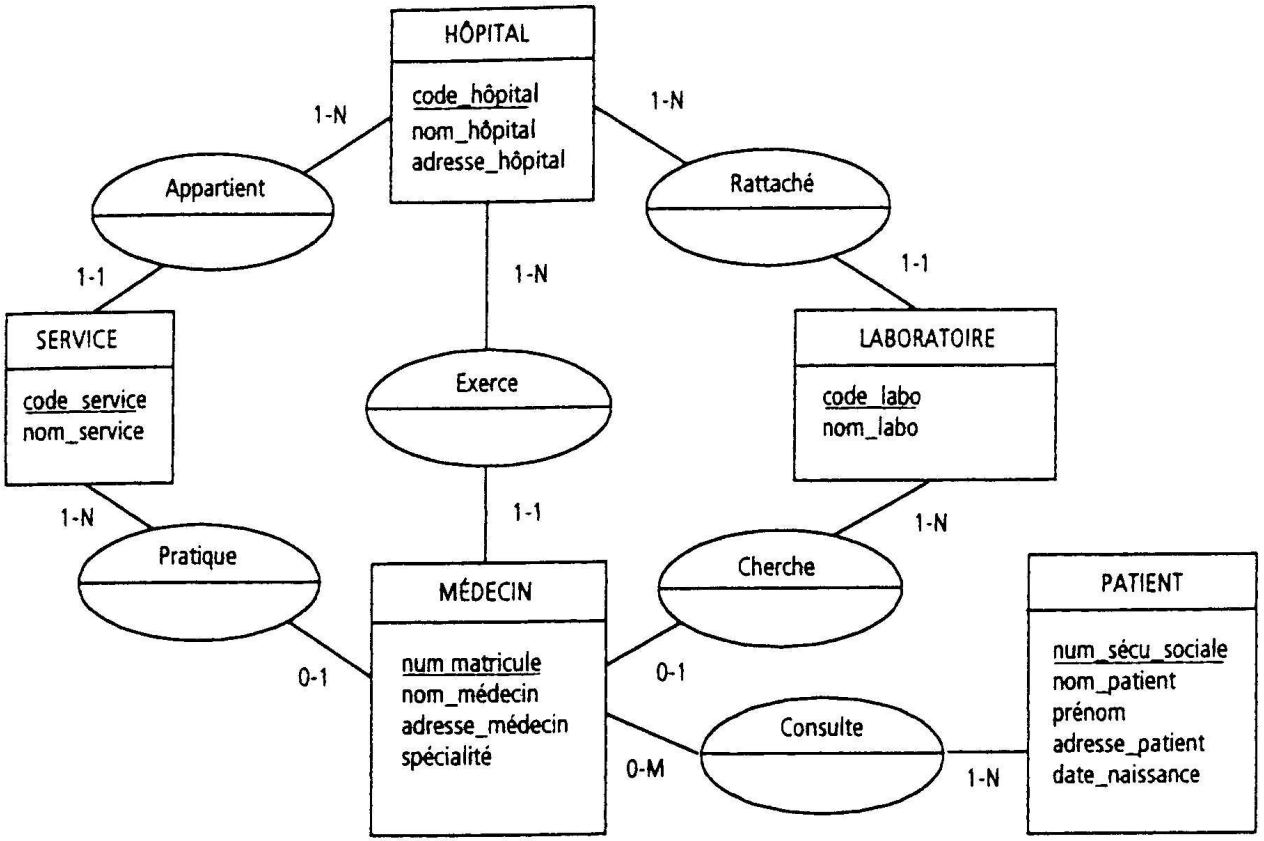

HÔPITAL
code_hôpital
nom_hôpital
adresse_hôpital
1-N
1-N
Appartient
Rattaché
1-1
1-N
1-1
SERVICE
code_service
nom_service
Exerce
LABORATOIRE
code_labo
nom_labo
1-N
1-1
1-N
Pratique
MÉDECIN
Cherche
PATIENT
num_sécu_sociale
nom_patient
prénom
adresse_patient
date_naissance
0-1
num matricule
nom_médecin
adresse_médecin
spécialité
0-1
Consulte
0-M
1-N

Schéma EER

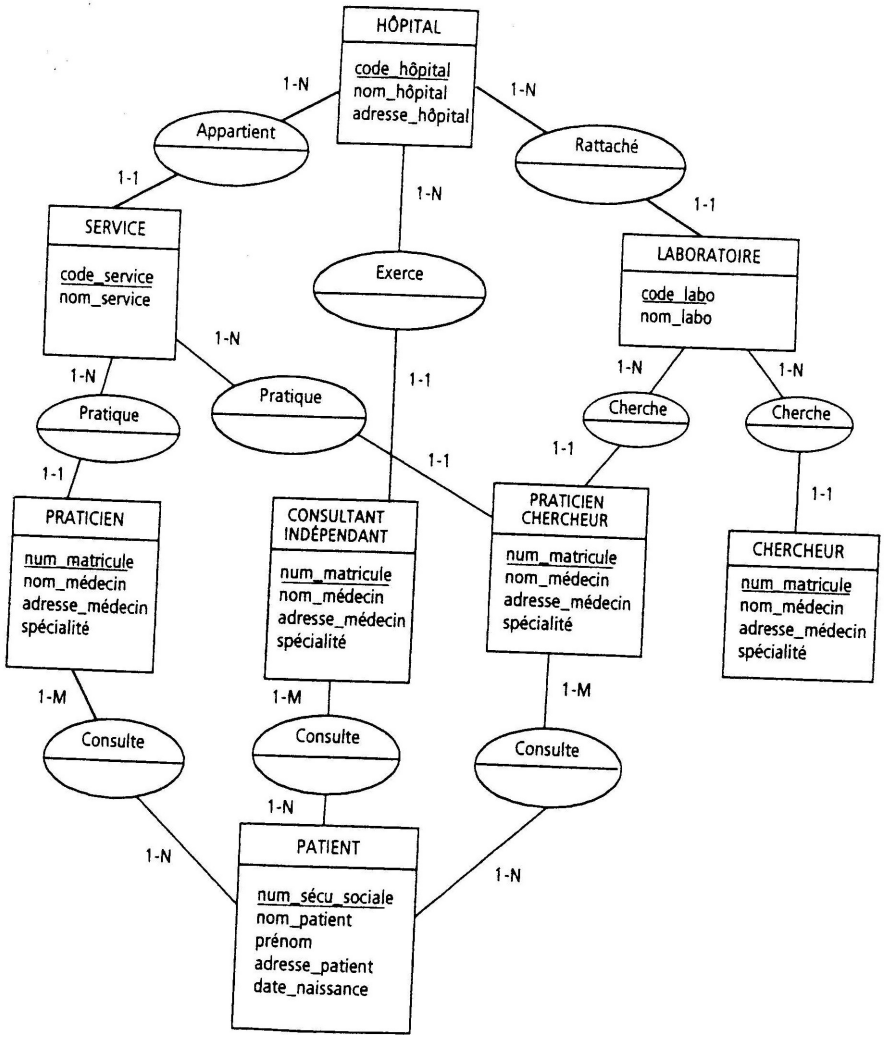
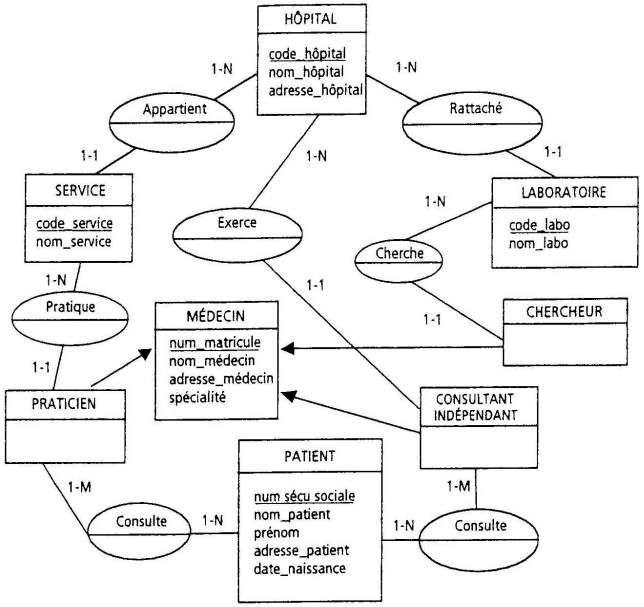
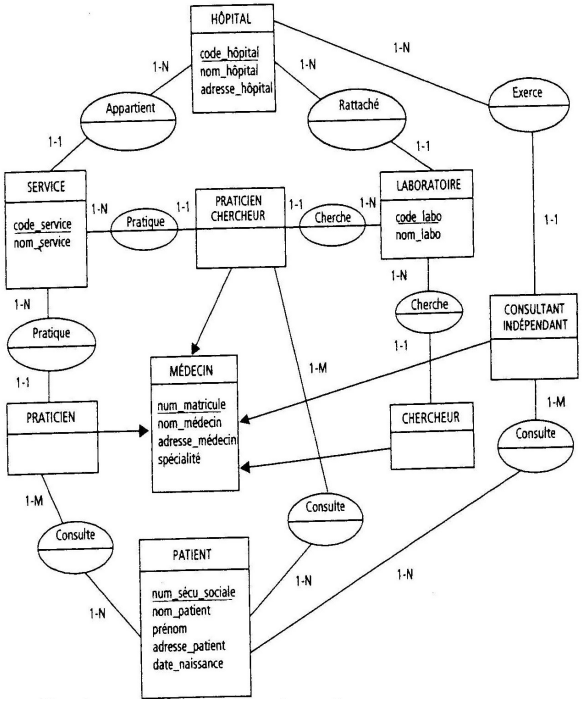


Schéma EER avec 4 catégories de médecins

Schéma EER avec généralisations





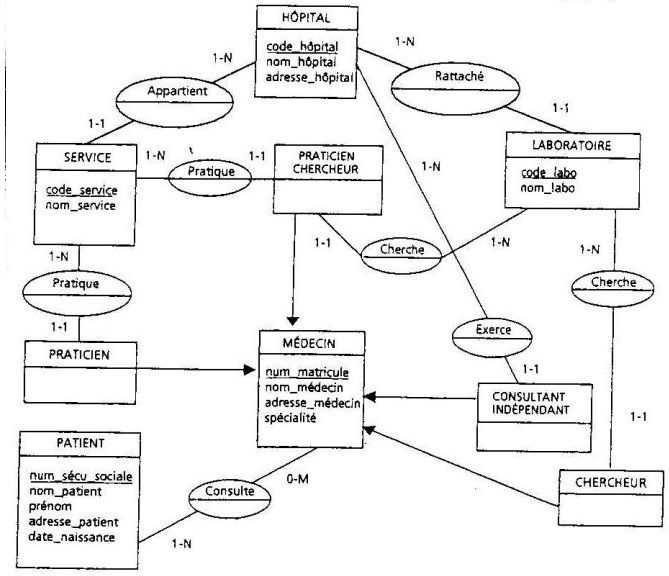
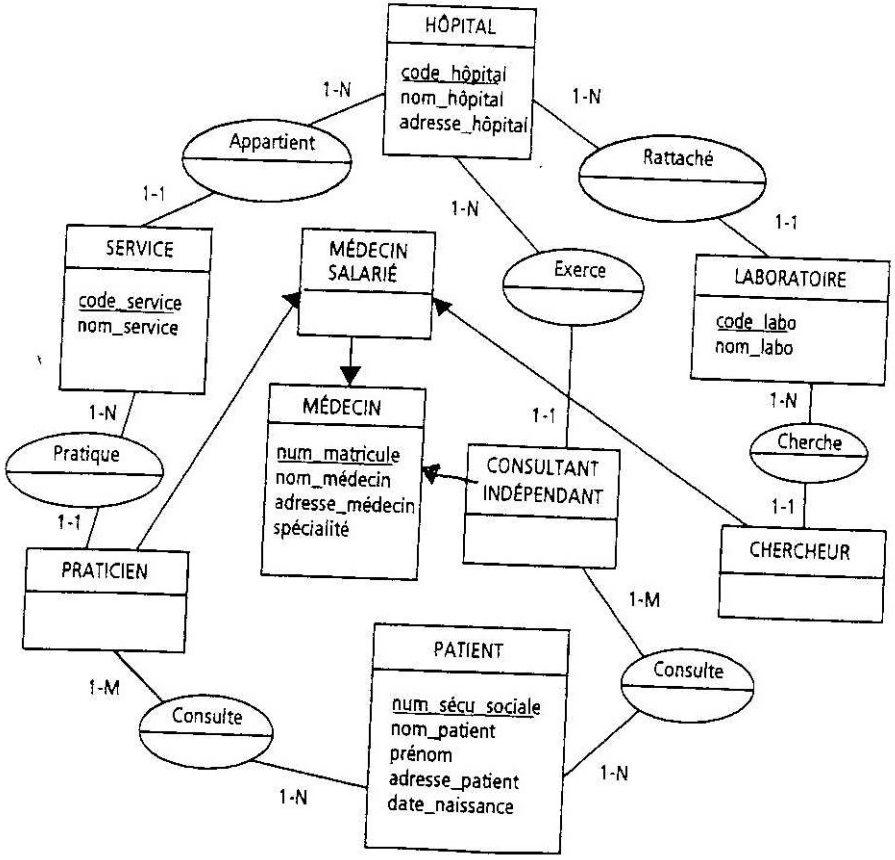
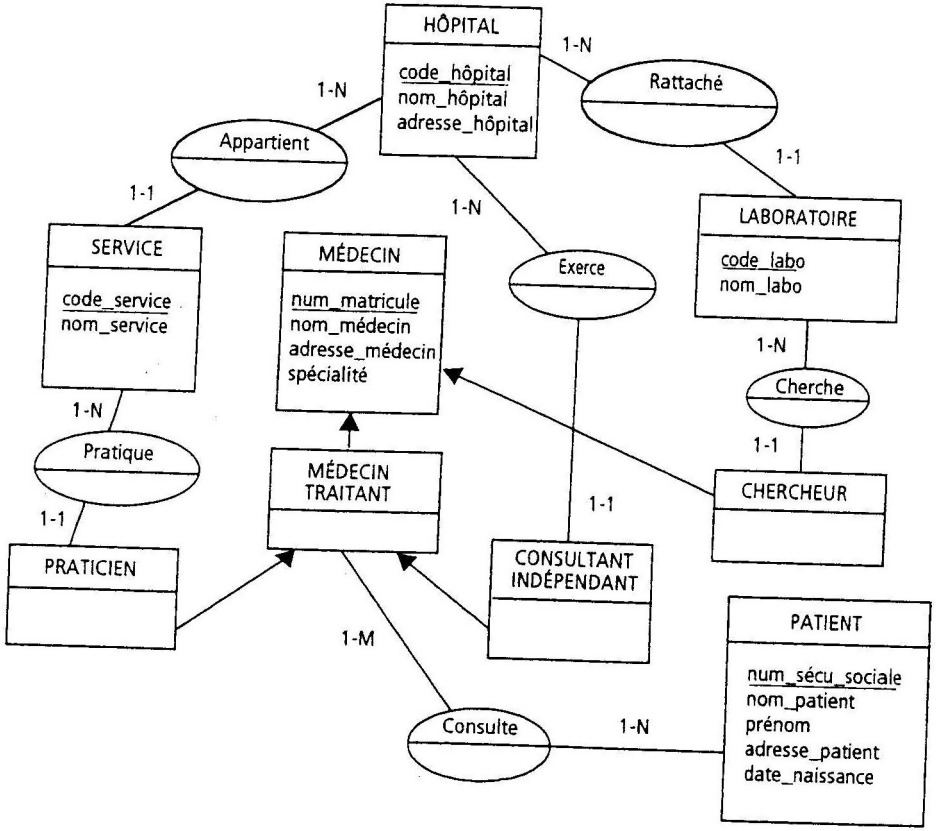


Schéma EER avec généralisations à 2 niveaux





Bibliographie

1. Bases de Données, Georges Gardarin, Ed. Eyrolles, Best Of Eyrolles, 19 euros en 2004 :
   ouvrage de référence en France, complet mais très orienté vers la théorie et finalement peu pédagogique.

3. Conception des bases de données relationnelles en pratique, J. AKOKA et I. COMYN, Ed. Vuibert Informatique :
   ouvrage très pratique avec exercices et corrigés.

Bases de Données

Nicolas Loménie

MODELE RELATIONNEL

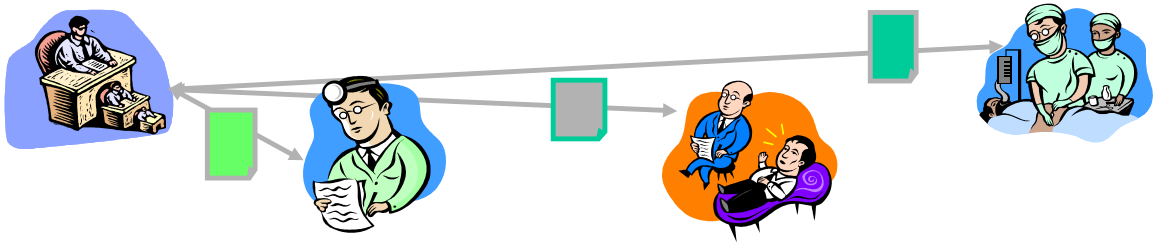
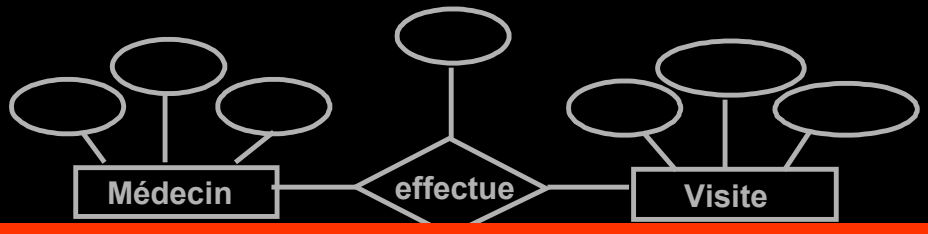
MODELE RELATIONNEL

Modéliser mathématiquement la BD

Objectifs :

- Apprendre les définitions essentielles des objets manipulés par le modèle relationnel : domaine, relation, schéma d'une relation, attribut et tuple, contrainte d'intégrités, clés primaire et étrangère, schéma relationnel.

- Etre capable d'appliquer la procédure de passage du schéma conceptuel (diagramme) au schéma relationnel (expression formelle).

Réel					
Modèle conceptuel	• Indépendant du modèle de données • Indépendant du SGBD				
Modèle logique	• Dépendant du modèle de données • Indépendant du SGBD	**Codasyl**	**Relationnel**	**Objet**	**XML**
Modèle Physique	• Dépendant du modèle de données • Dépendant du SGBD	• **Organisation physique des données** • **Structures de stockage des données** • **Structures accélératrices (index)**			

IMPLEMENTATION

- Nécessité de rendre **opérationnel** le schéma conceptuel à l'aide d'une représentation logique, par exemple relationnelle

- Conçue par E.F. Codd (1970s) : définition rigoureuse des concepts fondée sur la **théorie mathématique des relations, liée à celle des ensembles :** $\subset$, $\cup$, $\cap$

Domaine =
ensemble de valeurs défini en extension *(par exemple l'ensemble des grades d'un Salarié*) ou en *intention (par exemple le salaire d'un Salarié).*

Exemples:

- ENTIER
- REEL
- CHAINES DE CARACTERES
- EUROS
- SALAIRE = {4 000..100 000}
- COULEUR= {BLEU, BLANC, ROUGE}
- POINT = {(X:REEL,Y:REEL)}
- TRIANGLE = {(P1:POINT,P2:POINT,P3:POINT)}

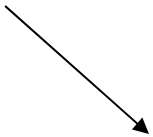
Notion de granularité de l'information: *voir l'exemple d'une date ou d'une adresse*.

Produit cartésien =

Le produit cartésien de plusieurs domaines $D_1 x D_2 x \ldots x D_n$ est l'ensemble des tuples (N-UPLETS) $<V_1, V_2, \ldots V_n>$ tels que $V_i \in D_i$

Exemple:

- D_1 = {Bleu,Blanc,Rouge}
- D_2 = {Vrai, Faux}
- $D_1 x D_2$ = {(Bleu,Vrai),(Bleu,Faux),(Blanc,Vrai),…}



Bleu	Vrai
Bleu	Faux
Blanc	Vrai
Blanc	Faux
Rouge	Vrai
Rouge	Faux

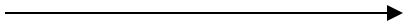
Relation ou Table =

sous–ensemble du produit cartésien d'une liste de domaines, non nécessairement tous distincts

= tableau à deux dimensions

- Exemple : la relation CoulVins issue du produit cartésien $D_1 x D_2$

$D_1 x D_2$	
Bleu	Vrai
Bleu	Faux
Blanc	Vrai
Blanc	Faux
Rouge	Vrai
Rouge	Faux



CoulVins	Coul	Choix
	Bleu	Faux
	Blanc	Vrai
	Rouge	Vrai

VISION TABULAIRE DU RELATIONNEL

- Une relation est une table à deux dimensions
- Une ligne est un tuple
- Un nom est associé à chaque colonne afin de la repérer indépendamment de son numéro d'ordre

Attribut ou Colonne =
sous-ensemble de valeurs d'un domaine

Tuple ou n-uplet ou t-uple ou Ligne =
ensemble de n valeurs prises dans les n domaines considérés

Degré d'une relation =
nombre de colonnes ou de domaines considérés

Schéma d'une relation =

nom de la relation suivi de la liste de ses attributs et de la définition de leurs domaines, par exemple la table *SALARIE* est caractérisé par le schéma relationnel suivant :

SALARIE(matricule : entier, nom : chaîne de caractères, grade : {employé, agent de maîtrise, cadre},salaire : [7000, 24000])

ou le plus souvent

SALARIE(matricule, nom, grade, salaire)

Schéma relationnel =

ensemble des schémas des relations d'une base de données ou ensemble des tables

Intégrité Logique ou Cohérence des données

->assurée par la vérification par le SGBD d'un ensemble de contraintes d'intégrité

->Objectif : Détecter les mises à jour erronée

-> **Avantages** :

- **simplification** du code des applications
- **sécurité renforcée** par l'automatisation
- **mise en commun** des contraintes

-> **Nécessite :**

- un langage de définition de contraintes d'intégrité
- la vérification **automatique** de ces contraintes

Contrainte d'intégrité =

expression logique qui doit être vraie, à tout moment, dans une base de données, *par exemple deux employés ne peuvent avoir le même matricule* ;

Exemples

- Contrôle sur les données élémentaires
  - Contrôle de types: Nom alphabétique
  - Contrôle de valeurs: Salaire mensuel entre 1 et 10kEuros

- Contrôle sur les relations entre les données
  - Relations entre données élémentaires:
    - Prix de vente > Prix d'achat
  - Relations entre objets:
    - Un électeur doit être inscrit sur une seule liste électorale

Contrainte de domaine

restriction de l'ensemble des valeurs possibles d'un domaine

Contrainte de clé

sous-ensemble minimal de colonnes tel que la table ne puisse contenir deux lignes ayant les mêmes valeurs pour ces colonnes

Contrainte obligatoire

un attribut doit toujours avoir une valeur

Contrainte d'intégrité référentielle ou d'inclusion notée $\subseteq$

elle lie deux ensemble de colonnes de deux tables différentes

Clé

Groupe d'attributs minimum qui détermine de façon unique un tuple dans une relation

Exemples:

- *NumSécuSociale* pour une PERSONNE

Contrainte d'entité

Toute relation doit posséder au moins une clé documentée

Clé Etrangère

Groupe d'attributs devant apparaître comme clé dans une autre relation

Exemples:

SALARIE(matricule, nom, grade, salaire, *Code_Filiale*)

FILIALE(Code_Filiale, nom, adresse)

Les clés étrangères définissent les contraintes d'intégrité référentielles

- Lors d'une insertion, la valeur des attributs doit exister dans la relation référencée
- Lors d'une suppression dans la relation référencée les tuples référençant doivent disparaître
- Elles correspondent aux liens entité-association obligatoires

Docteurs

Id-D	Nom	Prénom
1	Dupont	Pierre
2	Durand	Paul
3	Masse	Jean
….	……..	……

Visites

Id-D	Id-P	Id-V	Date	Prix
1	2	1	15 juin	250
1	1	2	12 août	180
2	2	3	13 juillet	350
2	3	4	1 mars	250

Prescriptions

Id-V	Ligne	Id-M	Posologie
1	1	12	1 par jour
1	2	5	10 gouttes
2	1	8	2 par jour
2	2	12	1 par jour
2	3	3	2 gouttes
….	….	….	……

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton
….	…….	…….	…….

Médicaments

Id-M	Nom	Description
1	Aspegic 1000	……..
2	Fluisédal	……..
3	Mucomyst	……..
….	……..	……..

Schéma d'une relation (bis) =
nom de la relation suivi de la liste des attributs, de la définition de leurs domaines et de l'ensemble des contraintes d'intégrité associées à cette table

exemple

SALARIE(matricule : entier, nom : chaîne de caractères, grade : {employé, agent de maîtrise, cadre},salaire : [7000, 24000])

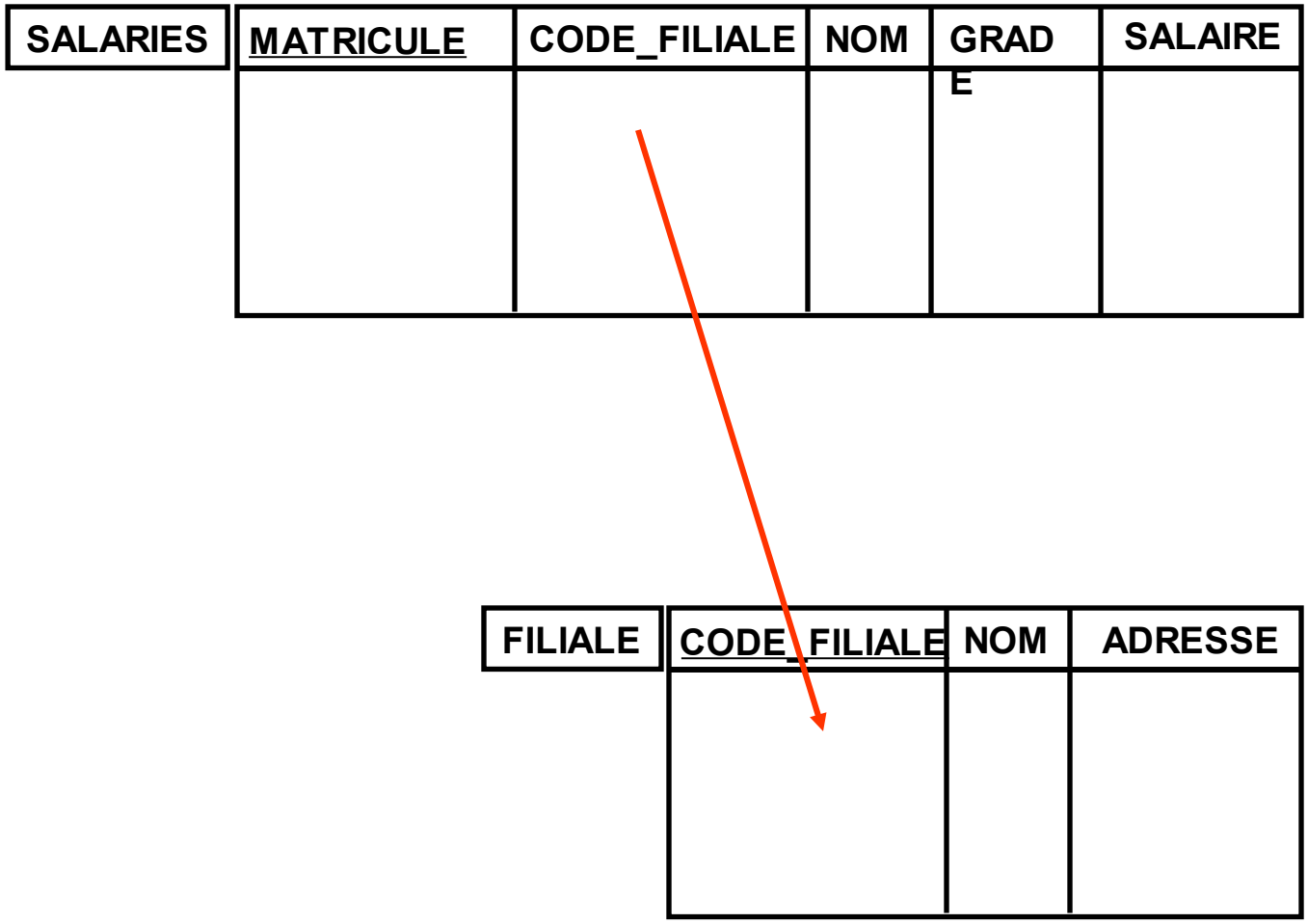
ou le plus souvent

SALARIE(matricule, nom, grade, salaire, *Code_Filiale*)

FILIALE(Code_Filiale, nom, adresse)

SALARIE[Code_Filiale] $\subseteq$ FILIALE[Code_Filiale]

Diagramme des Liens



Concepts Descriptifs : Bilan

- RELATION ou TABLE
- ATTRIBUT ou COLONNE
- DOMAINE ou TYPE
- CLE
- CLE ETRANGERE

INTENTION ET EXTENSION

- Un schéma de relation définit l'intention de la relation
- Une instance de table représente une extension de la relation

Théorie de la normalisation =
définition formelle de la qualité des tables au regard du problème posé par la redondance des données et au moyen de la notion de dépendance fonctionnelle

Dépendance fonctionnelle =
soit une table relationnelle T(A1,A2,…,An), soient X et Y deux sous-ensembles de colonnes de T, on dit que X détermine fonctionnellement Y (on note X->Y) si, à un n-uplet de valeurs de X, correspond au plus un n-uplet de valeurs de Y

Clé =
sous-ensemble minimal X de colonnes de T déterminant fonctionnellement toutes les colonnes de T

Forme Normale =

plus une table est normalisée, moins elle comporte de redondances et donc de risques d'incohérence :

1FN : la table ne contient que des attributs atomiques (code postal et ville);

2FN : 1FN + il n'existe pas de dépendance fonctionnelle entre une partie d'une clé et une colonne non clé de la table. Autrement dit toute colonne dépend de toute la clé.

3FN : 2FN + il n'existe aucune dépendance fonctionnelle entre les colonnes non clé

Algorithme de Bernstein =

algorithme permettant à partir de la seule connaissance des dépendances fonctionnelles entre attributs représentatives d'une situation de générer automatiquement un schéma relationnel en troisième forme normale

Une démarche possible MC -> MR

Règle 1. Toute entité est traduite en une table relationnelle dont les caractéristiques sont les suivantes :

- Le nom de la table est le nom de l'entité ;
- La clé de la table est l'identifiant de l'entité ;

Les autres attributs de l'entité forment les autres colonnes de la table.

HOPITAL(code_hopital, nom_hopital,adresse_hopital)

HOPITAL
code_hopital
nom_hopital
adresse_hopital

CREATE TABLE HOPITAL(code_hopital INT PRIMARY KEY, nom_hopital CHAR(10), adresse_hopital CHAR(30))

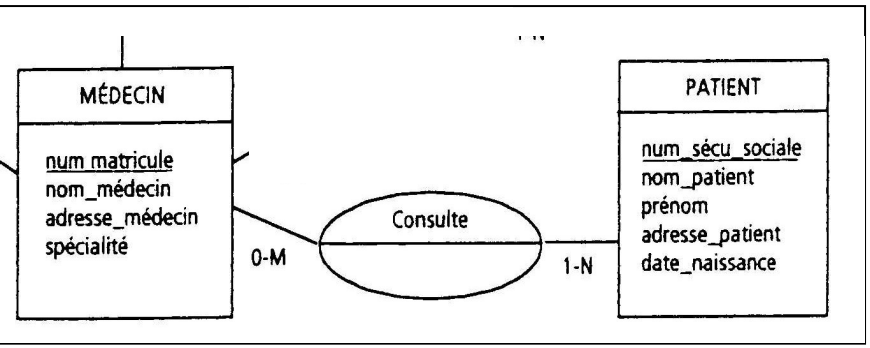
Règle 2. Toute relation binaire M-N est traduite en une table relationnelle dont les caractéristiques sont les suivantes :

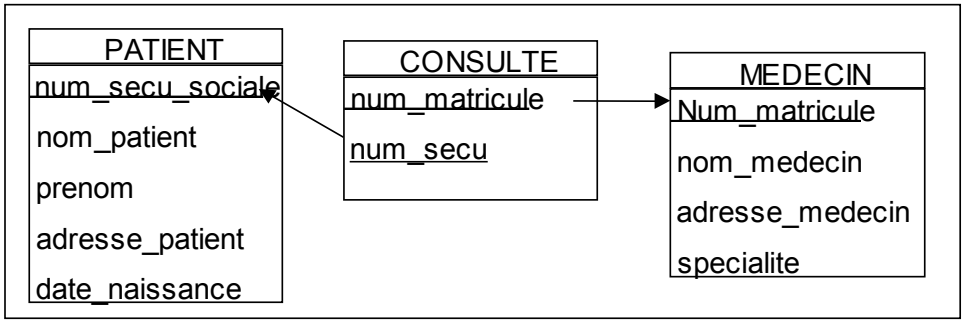
- Le nom de la table est le nom de la relation ;
- La clé de la table est formée par la concaténation des identifiants des entités participants à la relation ;
- Les attributs spécifiques de la relation forment les autre colonnes de la table ;
- Une contrainte d'intégrité référentielle est générée entre chaque colonne clé de la nouvelle table et la table d'origine ;

MEDECIN(num_matricule, nom_medecin,adresse_medecin,specialite)
PATIENT(num_secu_sociale, nom_patient,prenom,adresse_patient,date_naissance)

CONSULTE(num_matricule, num_secu)

CONSULTE[num_matricule] ⊆ MEDECIN[num_matricule]
CONSULTE[num_secu] ⊆ MEDECIN[num_secu_sociale]

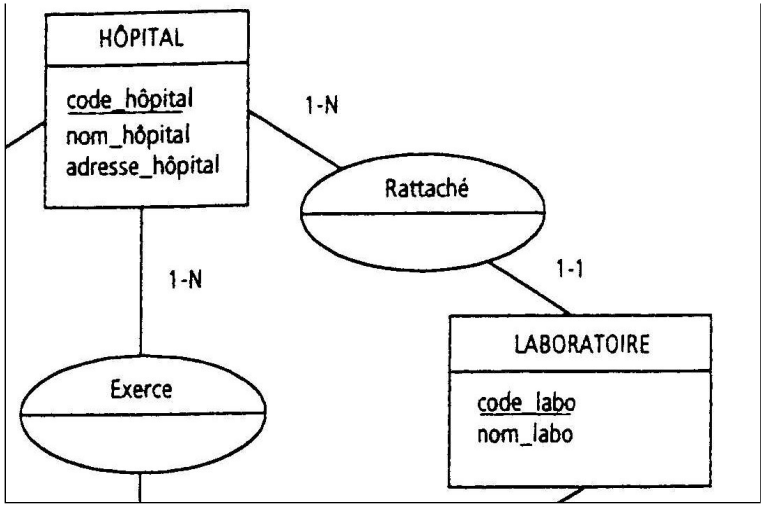


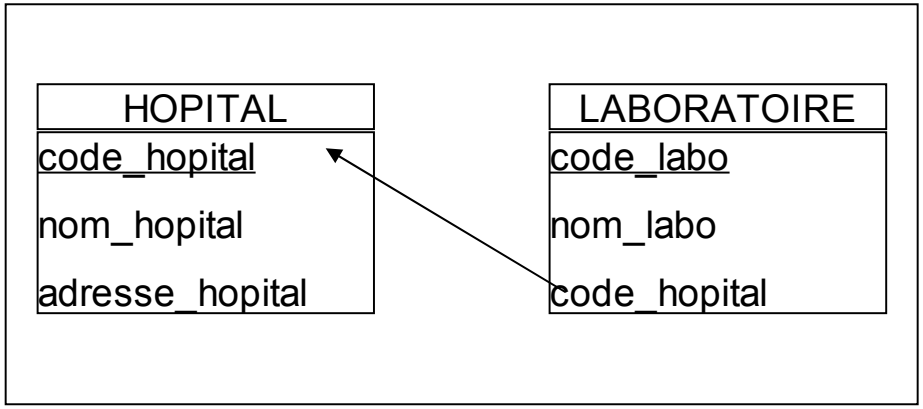


Règle 3. toute relation binaire I-N est traduite :

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

HOPITAL(code_hopital, nom_hopital,adresse_hopital)
LABORATOIRE(code_labo,nom_labo,code_hopital)





CREATE TABLE LABORATOIRE(code_labo INT PRIMARY KEY, nom_labo CHAR(10), code_hopital INT REFERENCES HOPITAL(code_hopital))

Règle 4. Toute relation binaire I-I est traduite, au choix, par l'une des trois solutions suivantes :

- Fusion des tables des entités qu'elle relie ;
- Report de clé d'une table dans l'autre ;
- Création d'une table spécifique reliant les clés des deux entités ;
- De plus, deux contraintes d'intégrité référentielle sont générées ;

Règle 5. Toute relation ternaire ou plus est traduite par une table spécifique ;

Règle 6. Toute relation récursive est considérée comme une relation binaire. Sa traduction dépend donc du type de cette relation binaire (I-I, I-N ou M-N) et obéit à l'une des règles 2,3 ou 4 ;

Règle 7. Toute généralisation E de n entités E1, E2, …, En peut être traduite au choix par l'une des trois solutions suivantes :

- La création d'une seule table représentant l'entité générique E et intégrant tous les attributs des entités spécifiques ;
- La création de n tables représentant les entités E1, E2,… En qui héritent de l'ensemble des attributs et des relations de l'entité générique E ;
- La création conjointe des tables E, E1, E2,… En reliées par des relations I-I (voir règle 4) ;
-

Règle 8. Vérifier le schéma relationnel obtenu au regard des dépendances fonctionnelles existant entre les attributs pour vérifier que l'ensemble des relations sont de type 3FN ;

Les erreurs à ne pas commettre :

- Sens du report de clé pour les relations 1-N ;
- Oubli de report des attributs spécifiques des relations 1-N ;
- Difficulté de l'identification des ternaires ou plus ;
- Répercussion des erreurs de modélisation conceptuelle ;

Ceci est une méthode, il en existe beaucoup d'autres qui parfois entreront en "contradiction" apparente avec celle-ci, notamment sur la gestion des cardinalités minimales pour le passage Conceptuel-> Relationnel.

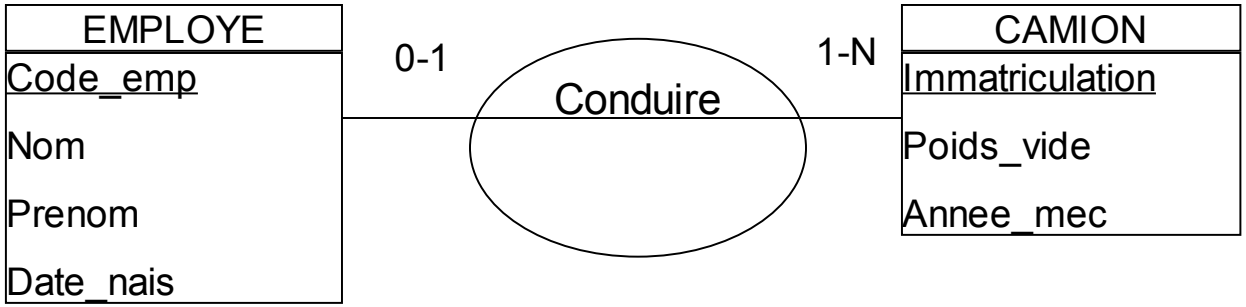
Constituer un bon schéma conceptuel est une affaire avant tout de bons sens. Si le schéma conceptuel est cohérent, on obtient sans aucun doute un schéma relationnel au niveau minimum de la troisième forme normale.

La théorie de la normalisation est surtout utile quand on reprend une base qui existe déjà et qu'il faut essayer de la "réparer". Dans ce cas, faire la liste des dépendances fonctionnelles et appliquer l'algorithme de type Bernstein peut être une solution pour retrouver la cohérence.

Essayer de retrouver le modèle conceptuel est aussi une façon de procéder (rétroengineering).

Dans tous les cas, ce n'est pas une tache facile.

D'autant plus, que pour des raisons d'optimisation (en temps de réponse essentiellement), on est parfois amené à dénormaliser un schéma en créant des redondances et au prix d'un code un peu plus lourd.



A cause de la cardinalité 0-1, deux possibilités :

• On la traite comme une relation 1-N :

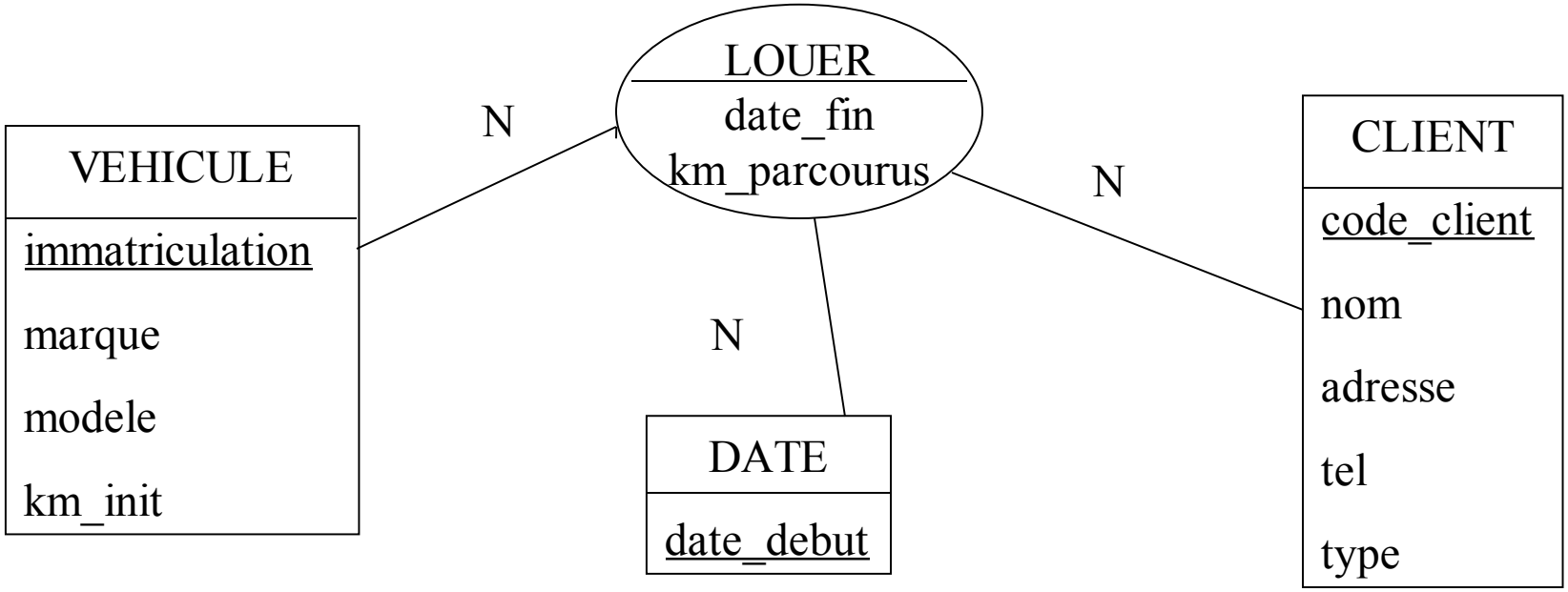
EMPLOYE(Code_emp,Nom,Prenom,Date_nais,Immatriculation)
CAMION(Immatriculation, Poids_vide,Annee_mec)

•On la traite comme une relation N-N, pour éviter les valeurs nulles dans EMPLOYE pour le champ Immatriculation

EMPLOYE(Code_emp,Nom,Prenom,Date_nais)
CAMION(Immatriculation, Poids_vide,Annee_mec)
CONDUIRE(Code_emp,Immatriculation)

Du coup, en analysant les dépendances fonctionnelles dans CONDUIRE on s'aperçoit que Code_emp -> Immatriculation, et pour être en troisième forme normale il faut écrire CONDUIRE (Code_emp, Immatriculation). Vous vous apercevez ici que passer du MR au MCD n'est pas une tache aisé, étant donné que les méthodes de passage du MCD au MR peuvent légèrement différer en fonction des pratiques de chaque concepteur.

Ce MCD



devient :

VEHICULE(immat, marque, modèle, km_init)
CLIENT(code_client,nom, adresse, tel, type)
LOUER(immat,code_client,date_debut,date_fin,km_parcouru)

où on a supprimé le schéma de la relation DATE car il ne contient qu'un attribut (sa clé)

Bases de Données

Nicolas Loménie

ALGEBRE RELATIONNELLE

ALGEBRE RELATIONNELLE

Langage de requêtes

- Fondement mathématique à la base de SQL : le langage des Bases de Données

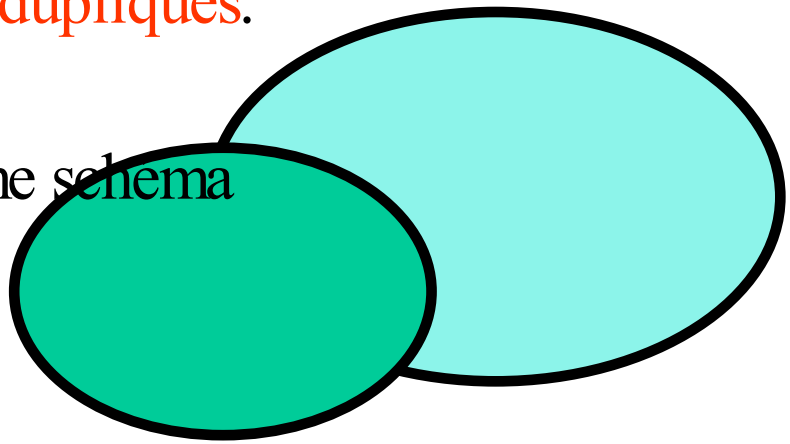
- Traduction **automatique** des questions déclaratives *(qui sont les employés habitant à Paris ?)* en programmes procéduraux.

- Optimisation **automatique** des questions

  - Langage opérationnel, très utile pour représenter les plans d'exécution.

Opérations Ensemblistes

- Ensemble : une définition mathématique pour une collection d'objets ne possédant pas d'éléments dupliqués.



- Opérations pour des relations de même schema
  - UNION notée $\cup$
  - INTERSECTION notée $\cap$
  - DIFFERENCE notée $-$
- Opérations binaires pour des relations de schémas différents
  - PRODUIT CARTESIEN : Relation X Relation --> Relation
  - JOINTURE : Relation $\bowtie$ Relation --> Relation
  - DIVISION : Relation $/$ Relation --> Relation
- Opérations algébriques sur une relation
  - PROJECTION notée π
  - RESTRICTION notée σ

Projection

- Elimination des attributs non désirés et suppression des tuples en double;
- Algorithme de tris !! donc très coûteux en temps de calcul;
- *Relation -> Relation* notée:

  $\pi_{A1,A2,...Ap}(R)$
- Avec A_i pour i de 1 à n une liste de domaines

VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C

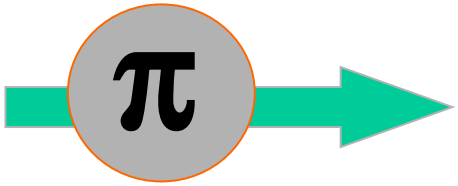
$\pi_{Cru,Région}$

$\pi_{Cru,Région}$(VINS)	Cru	Région
	VOLNAY	BOURGOGNE
	CHENAS	BEAUJOLAIS
	JULIENAS	BEAUJOLAIS

Projection

Patients

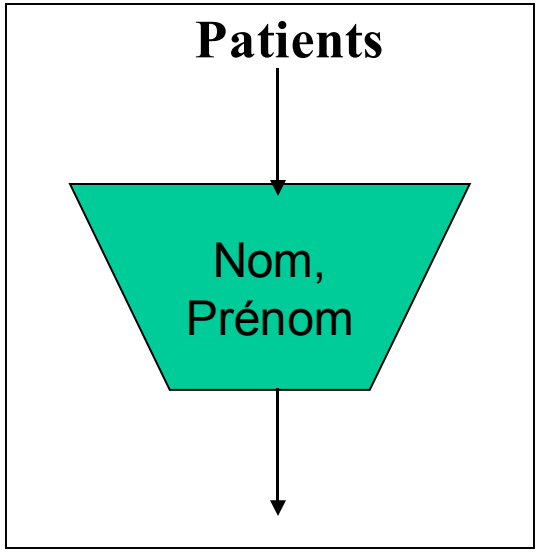
Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton



Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton

Nom et prénom des patients ?



Restriction

- Obtention des tuples de R satisfaisant un critère Q
- *Relation ->Relation*, notée $\sigma_Q(R)$
- Q est le critère de qualification de la forme :
  - (A_i Op Valeur) avec Op ={ =, <, >=, >, <=, !=}
  - Il est possible de réaliser des "ou" (union) et des "et" (intersection) de critères simples

VINS	Cru	Mill	Région	Qualité
	VOLNAY	1983	BOURGOGNE	A
	VOLNAY	1979	BOURGOGNE	B
	CHENAS	1983	BEAUJOLAIS	A
	JULIENAS	1986	BEAUJOLAIS	C

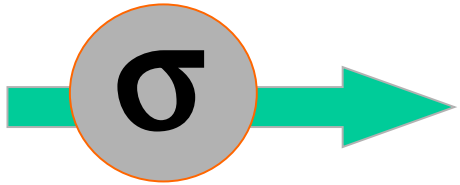
$\sigma_{MILL>1983}$

VINS	Cru	Mill	Région	Qualité
	JULIENAS	1986	BEAUJOLAIS	C

Restriction (ou Sélection)

Patients

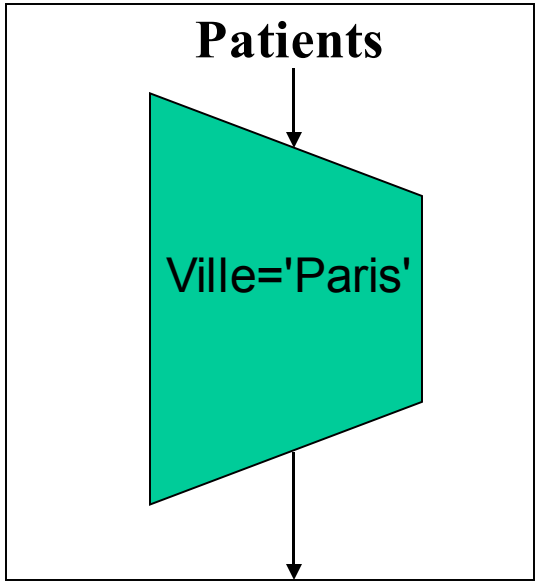
Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton



Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
3	Doe	John	Paris

Patients habitant Paris ?



Produit cartésien

- Composition de deux relations sur un domaine commun;
- = Cross Produit

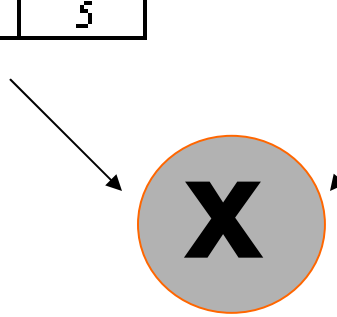
- *Relation*² ->*Relation*, notée $R_1 x R_2$

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4



R CROSS S

A	1	A	1
A	1	C	2
A	1	D	3
A	1	E	4
B	2	A	1
B	2	C	2
B	2	D	3
B	2	E	4
D	3	A	1
D	3	C	2
D	3	D	3
D	3	E	4

F	4	A	1
F	4	C	2
F	4	D	3
F	4	E	4
E	5	A	1
E	5	C	2
E	5	D	3
E	5	E	4

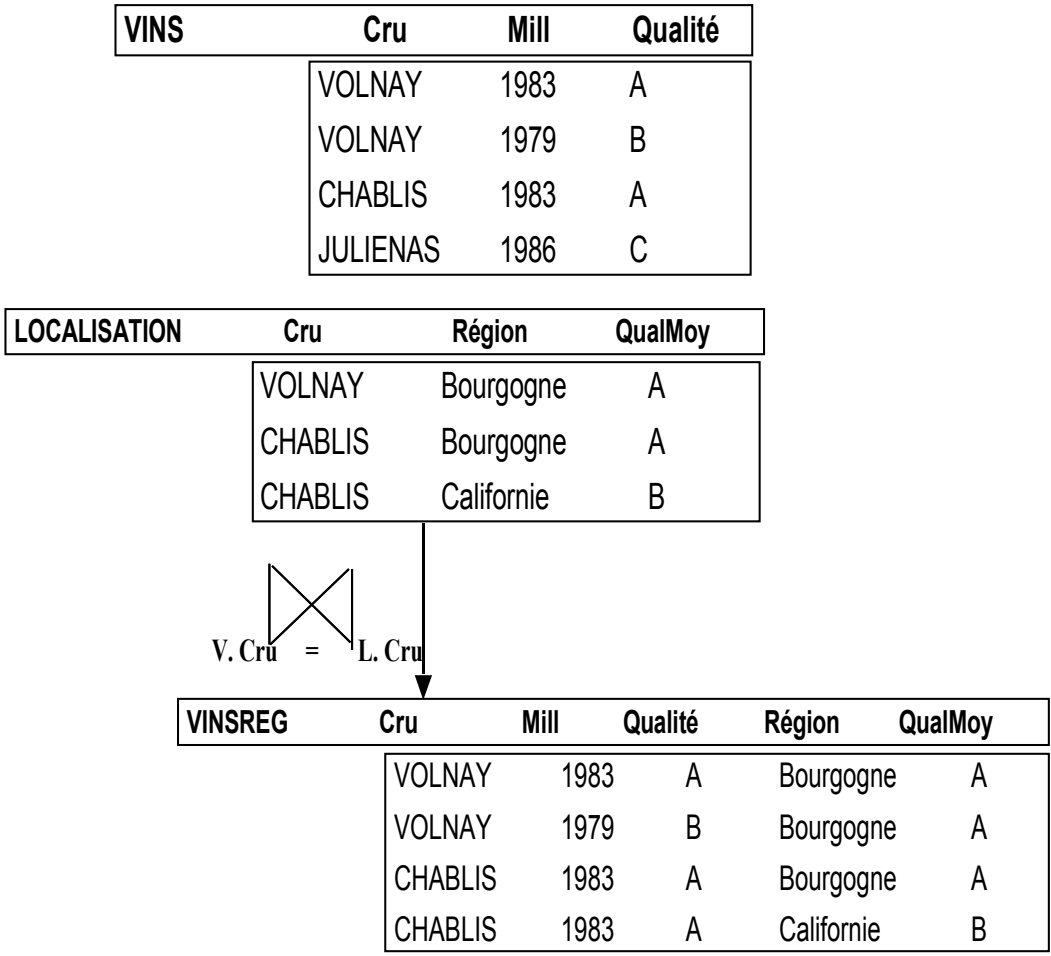
Jointure

- Composition de deux relations sur un domaine commun AVEC UNE CONDITION de jointure !

- *Relation² ->Relation*, notée $R_1 \underset{Q}{\bowtie} R_2$

- Critère de jointure Q,
  - Attributs égaux :
    - Attribut1 = Attribut2
    - Equi-jointure
  - Comparaison d'attributs :
    - Attribut1 θ Attribut2
    - Théta-jointure

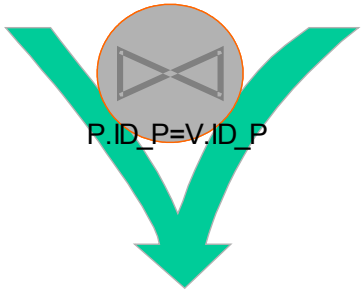
- Sous-ensemble du produit cartésien des deux relations



Jointure

Patients

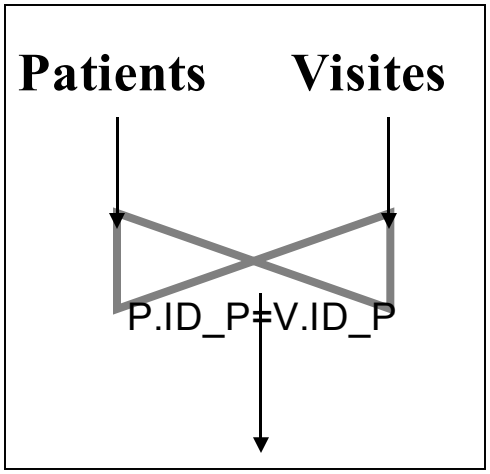
Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton



Visites

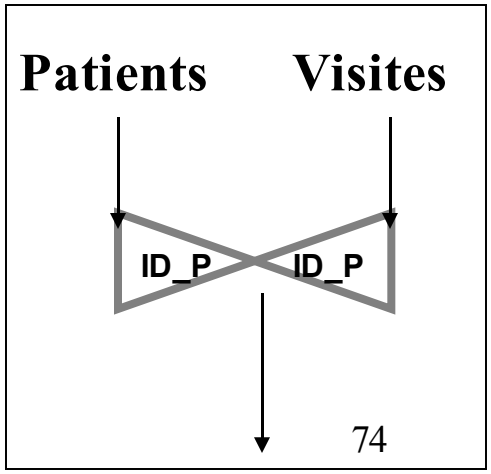
Id-D	Id-P	Id-V	Date	Prix
1	2	1	15 juin	250
1	1	2	12 août	180
2	2	3	13 juillet	350
2	3	4	1 mars	250

Id-P	Nom	Prénom	Ville	Id-D	Id-P	Id-V	Date	Prix
1	Lebeau	Jacques	Paris	1	2	1	12 août	180
2	Troger	Zoe	Evry	1	2	1	15 juin	250
2	Troger	Zoe	Evry	2	2	3	13 juillet	350
3	Doe	John	Paris	2	3	4	1 mars	250



En fait c'est une équi-jointure

Qu'on peut représenter ainsi sans ambiguïté

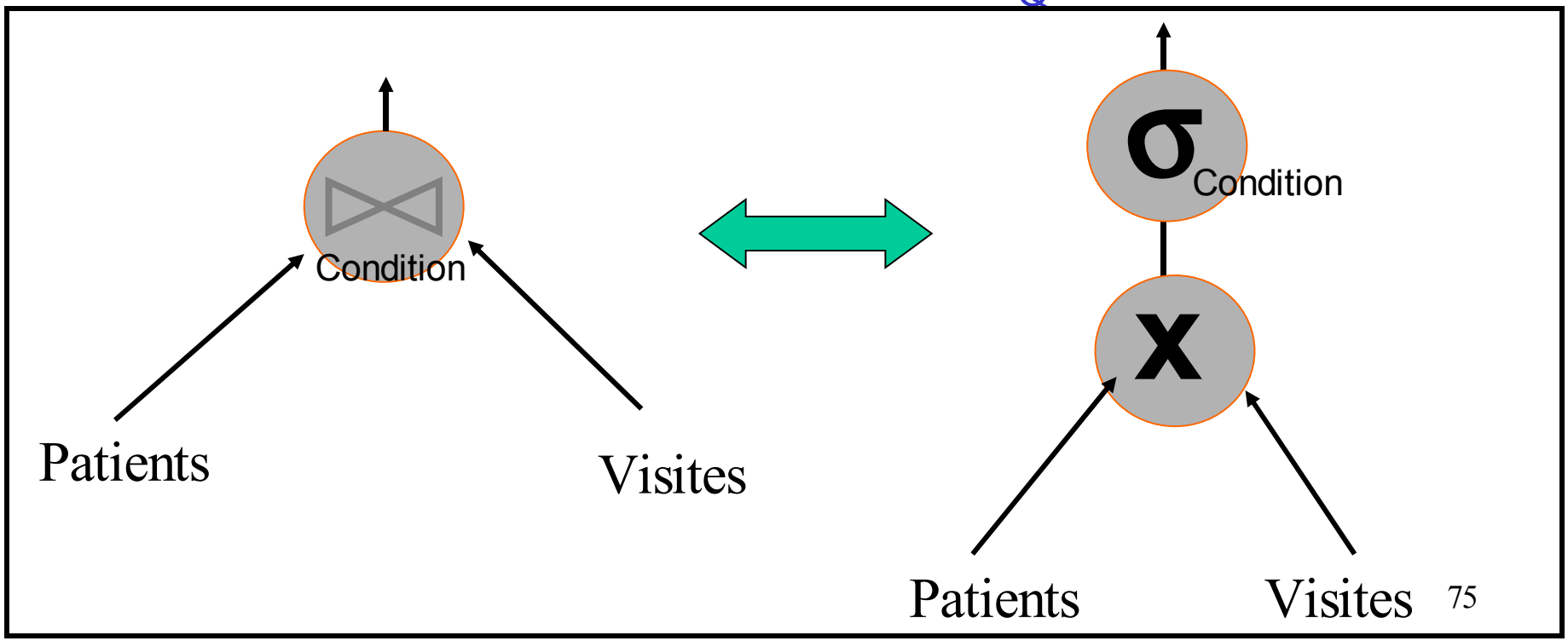


Jointure :

Une jointure est une composition d'un PRODUIT CARTESIEN et d'une RESTRICTION.

Attention : R ⋈ S = RxS !! Très dangereux !

En conséquence, c'est assez coûteux en place mémoire et en temps de calcul mais R $\bowtie_Q$ S ≠ RxS.



OUTER JOINs

Remarquez que la plupart des infos sont perdues en appliquant une jointure à deux relations.

Les jointures externes sont utilisées pour afficher tous les n-uplets, y compris ceux n'ayant pas de correspondance dans l'une ou l'autre des relations concernées.

Une jointure externe conserve les informations qui auraient été perdues, en remplaçant les valeurs manquantes avec la valeur NULL.

Il existe 3 formes de jointure externe, dépendant des données que l'on veut conserver.

- LEFT OUTER JOIN - keep data from the left-hand table
- RIGHT OUTER JOIN - keep data from the right-hand table
- FULL OUTER JOIN - keep data from both tables

OUTER JOIN Exemple 1

R

ColA	*ColB*
A	1
B	2
D	3
F	4
E	5

S

SColA	*SColB*
A	1
C	2
D	3
E	4

R LEFT OUTER JOIN$_{R.ColA = S.SColA}$S

A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-

R RIGHT OUTER JOIN$_{R.ColA = S.SColA}$S

A	1	A	1
D	3	D	3
E	5	E	4
-	-	C	2

OUTER JOIN Exemple 2

R

ColA	*ColB*
A	1
B	2
D	3
F	4
E	5

S

SColA	*SColB*
A	1
C	2
D	3
E	4

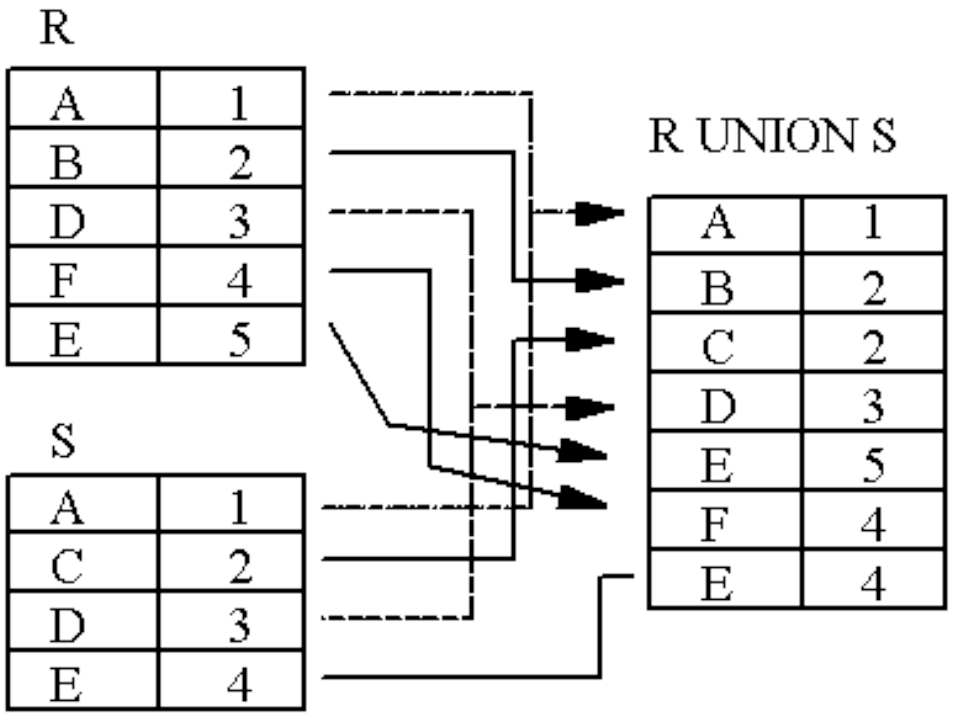
R FULL OUTER JOIN S R.ColA = S.SColA

A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-
-	-	C	2

Symbole ?

Union

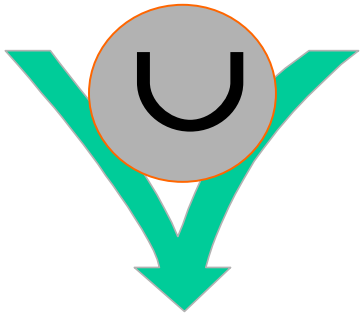
- Union ensembliste de deux relations de MEME SCHEMA !
- *Relation*² ->*Relation*, notée $R_1 \cup R_2$



UNION

Patients

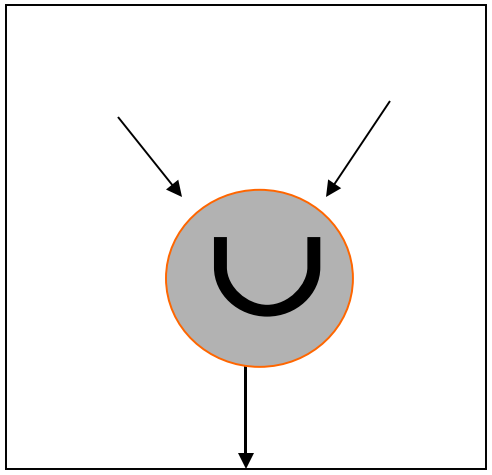
	Nom	Prénom	Ville
	Lebeau	Jacques	Paris
	Troger	Zoe	Evry
	Doe	John	Paris
	Perry	Paule	Valenton



Malades

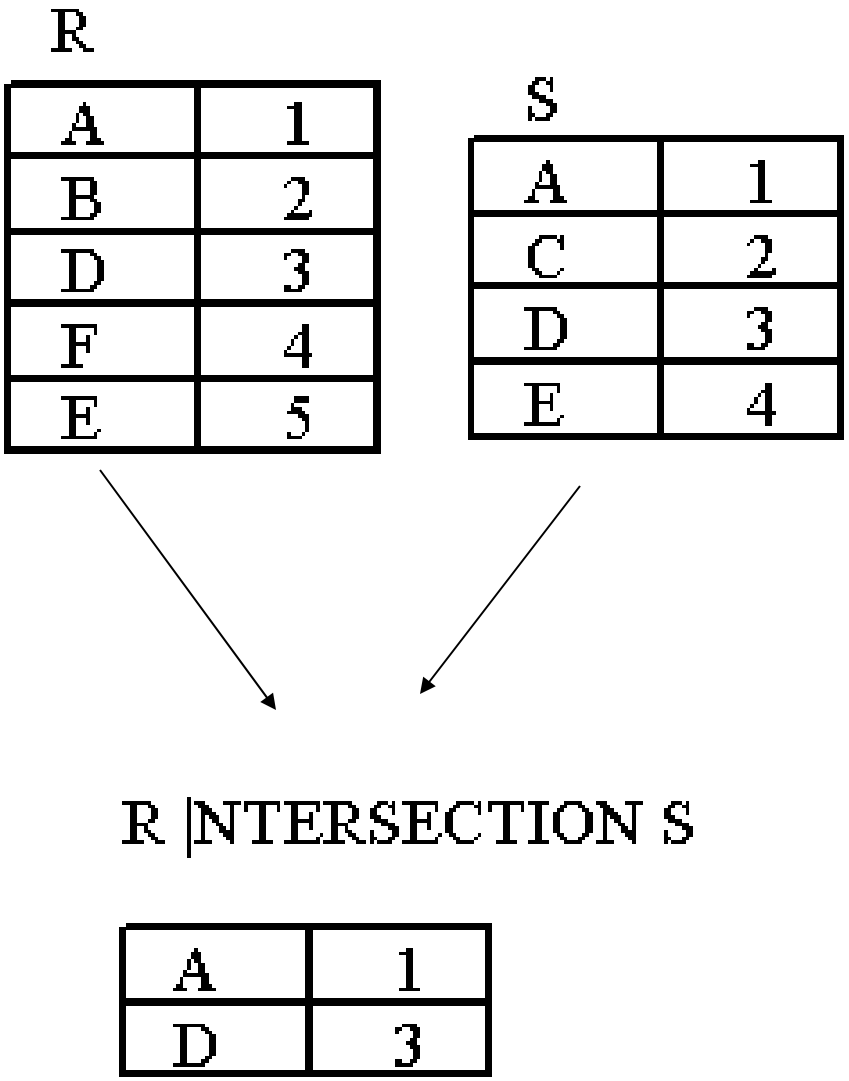
	Nom	Prénom	Ville
	Ducq	Pierre	Paris
	Lebeau	Jacques	Paris
	Foch	Didier	Paris

Lebeau	Jacques	Paris
Troger	Zoe	Evry
Doe	John	Paris
Perry	Paule	Valenton
Ducq	Pierre	Paris
Foch	Didier	Paris



Intersection

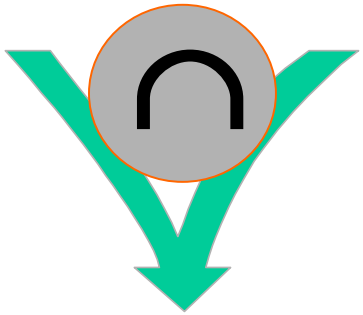
- Intersection ensembliste de deux relations de MEME SCHEMA !
- *Relation*² ->*Relation*, notée $R_1 \cap R_2$



INTERSECTION

Patients

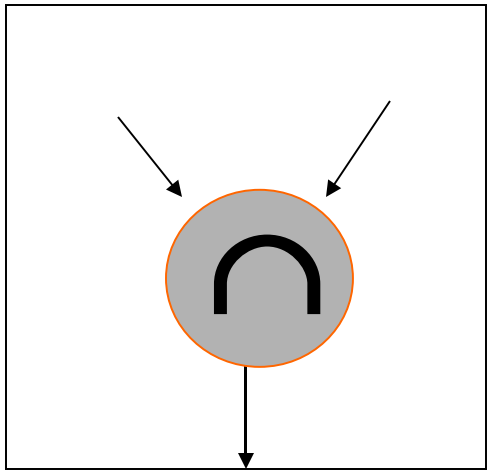
Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton



Malades

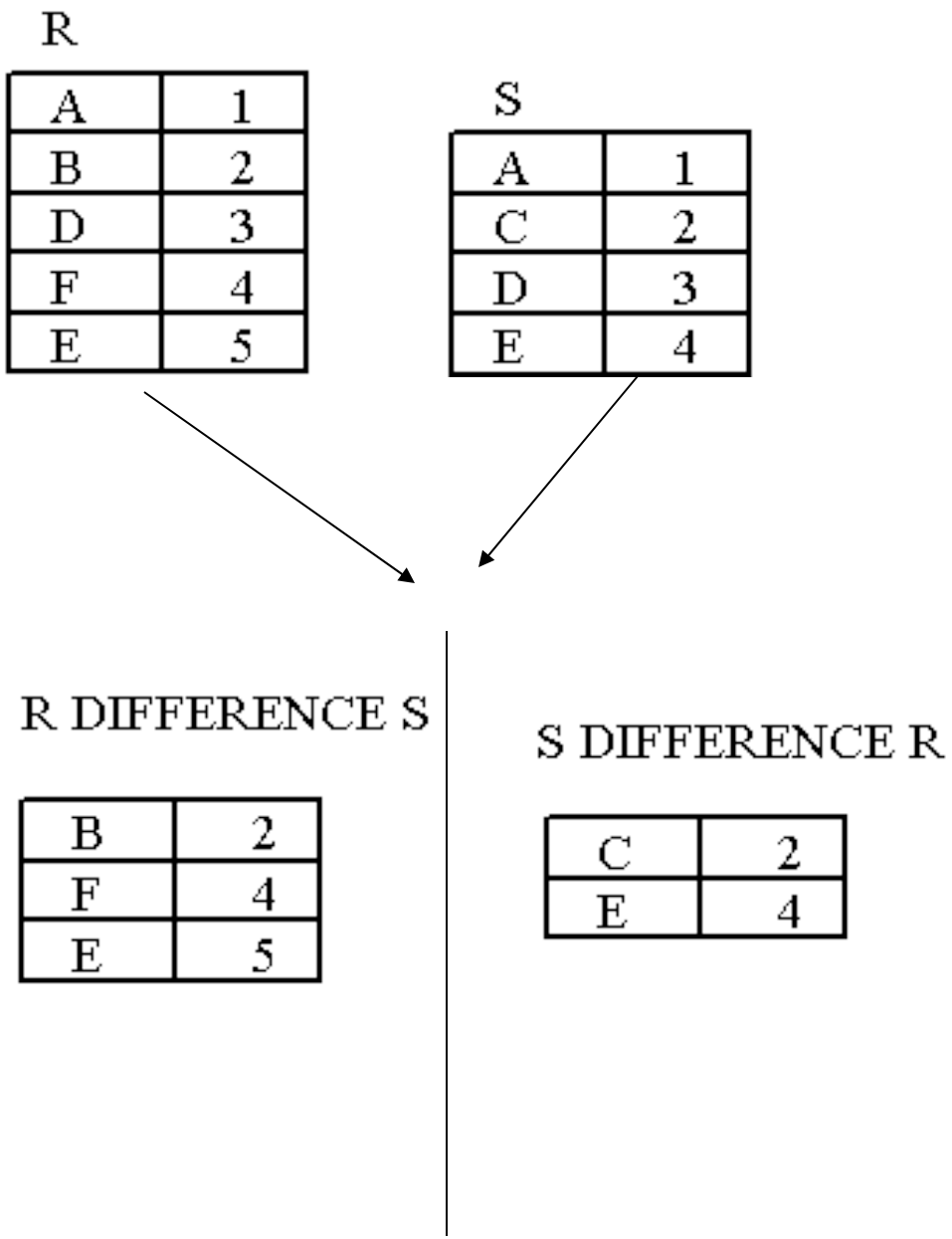
Id-M	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Ducq	Pierre	Paris
3	Foch	Didier	Paris

1	Lebeau	Jacques	Paris



DIFFERENCE

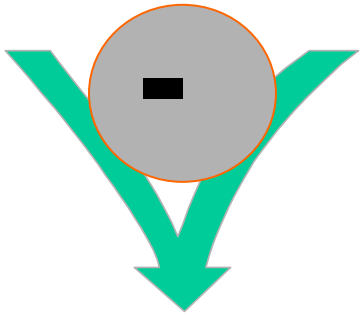
- Différence ensembliste de deux relations de MEME SCHEMA !
- *Relation*² ->*Relation*, notée $R_1 - R_2$
- Attention $R_1 - R_2 \neq R_2 - R_1$



DIFFERENCE

Patients

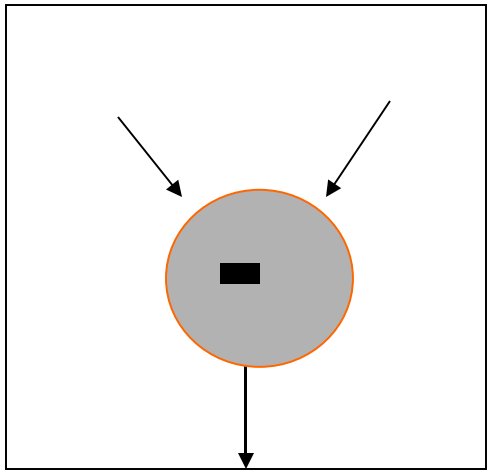
	Nom	Prénom	Ville
	Lebeau	Jacques	Paris
	Troger	Zoe	Evry
	Doe	John	Paris
	Perry	Paule	Valenton



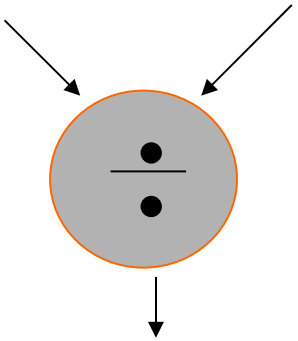
Malades

	Nom	Prénom	Ville
	Ducq	Pierre	Paris
	Lebeau	Jacques	Paris
	Foch	Didier	Paris

Troger	Zoe	Evry
Doe	John	Paris
Perry	Paule	Valenton



Division



- $R = R_1 / R_2$
- R(X) est la division de R_1 (X, Y) par R_2 (Y) ssi R contient tous les tuples (x) tels que $\forall(y) \in R_2, \exists(x, y) \in R_1$

Quels sont les fournisseurs de toutes les pièces ?

S#	P#
S1	P1
S1	P2
S2	P1

R_1

P#
P1
P2

R_2

S#
S1

R

Résumé

- Opérations de base:
  - Sélection (σ) : RESTRICT(Critère, Relation) ou SELECT
    - Sélectionne un sous-ensemble des lignes d'une relation.
  - Projection (π) : PROJECT(A_1,…,A_n, Relation)
    - Efface des colonnes d'une relation [et élimine les doubles].
  - Produit Cartésien (X)
    - Permet de combiner deux relations : PRODUIT(Relation1, Relation2)
  - Différence (-)
    - Elimine les tuples de R1 contenus dans R2
  - Union (∪)
    - Constitue une relation R avec les tuples de R1 et ceux de R2

- Opérations additionnelles:
  - Jointure (|¨ |) : JOIN(Critère, Relation1, Relation2)
    - Combinaison de produit cartésien et sélection sur colonne Θ comparables (=, <, >, ...)
  - Intersection
    - Constitue une relation R avec les tuples appartenant à la fois à R1 et R2
  - Division

Résumé (bis)

Il y a deux types d'opérateurs de requêtes :

- Fondés sur la théorie des ensembles :
  - ❖ UNION, INTERSECTION, DIFFERENCE, et CARTESIAN PRODUCT (ou CROSS PRODUCT ou JOIN PRODUCT)
- Spécifiques aux bases de données :
  - ❖ SELECT (pas le même que SQL SELECT), PROJECT, et JOIN.

- Chaque opération retournant une relation, les opérations peuvent être composées !

- L'algèbre relationnelle est fermée.

- L'algèbre relationnelle est complète
  - Les cinq opérations de base permettent de formaliser sous forme d'expressions toutes les questions que l'on peut poser avec la logique du premier ordre (sans fonction).

- Exemple :
  - Nom et prénom des buveurs de volnay 1988 ?

```
PROJECT (NOM, PRENOM,
   RESTRICT(CRU="VOLNAY" et MILL =1988,
      JOIN(VINS, ABUS, BUVEURS)))
```

SQL

- Une requête SQL est une paraphrase d'une expression de l'algèbre relationnelle en anglais;

- Requête élémentaire :

  SELECT A1, A2, …Ap
  FROM R1, R2, …Rk
  WHERE Q [{UNION |INTERSECT | EXCEPT } …]

- Sémantique du bloc select :

  PROJECT (A1,A2,…Ap, (
  RESTRICT (Q,
  PRODUIT (R1, R2, …, Rk))))

Le SELECT relationnel

SELECT is used to obtain a subset of the tuples of a relation that satisfy a *select condition*.

For example, find all employees born after 1st Jan 1950:

$$\text{SELECT}_{\text{dob> '01/JAN/1950'}}(\text{employee})$$

Le PROJECT relationnel

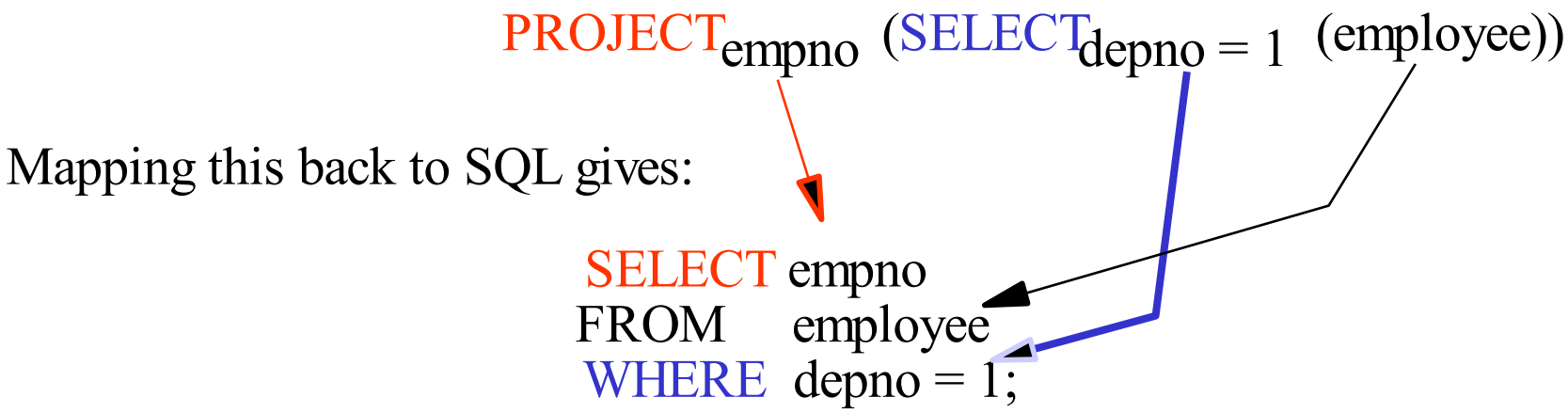
The PROJECT operation is used to select a subset of the attributes of a relation by specifying the names of the required attributes.

For example, to get a list of all employees surnames and employee numbers:

$$\text{PROJECT}_{\text{surname,empno}}(\text{employee})$$

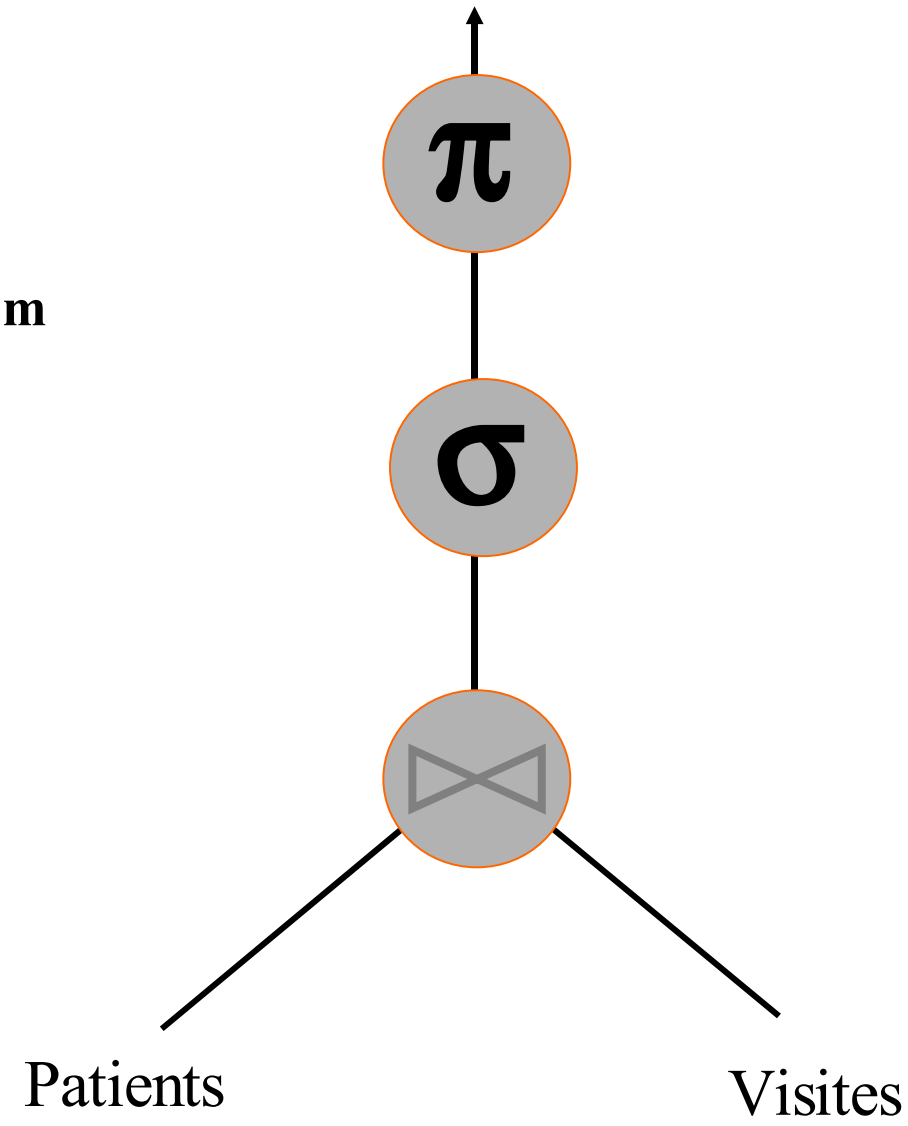
SELECT and PROJECT

SELECT and PROJECT can be combined together. For example, to get a list of employee numbers for employees in department number 1:

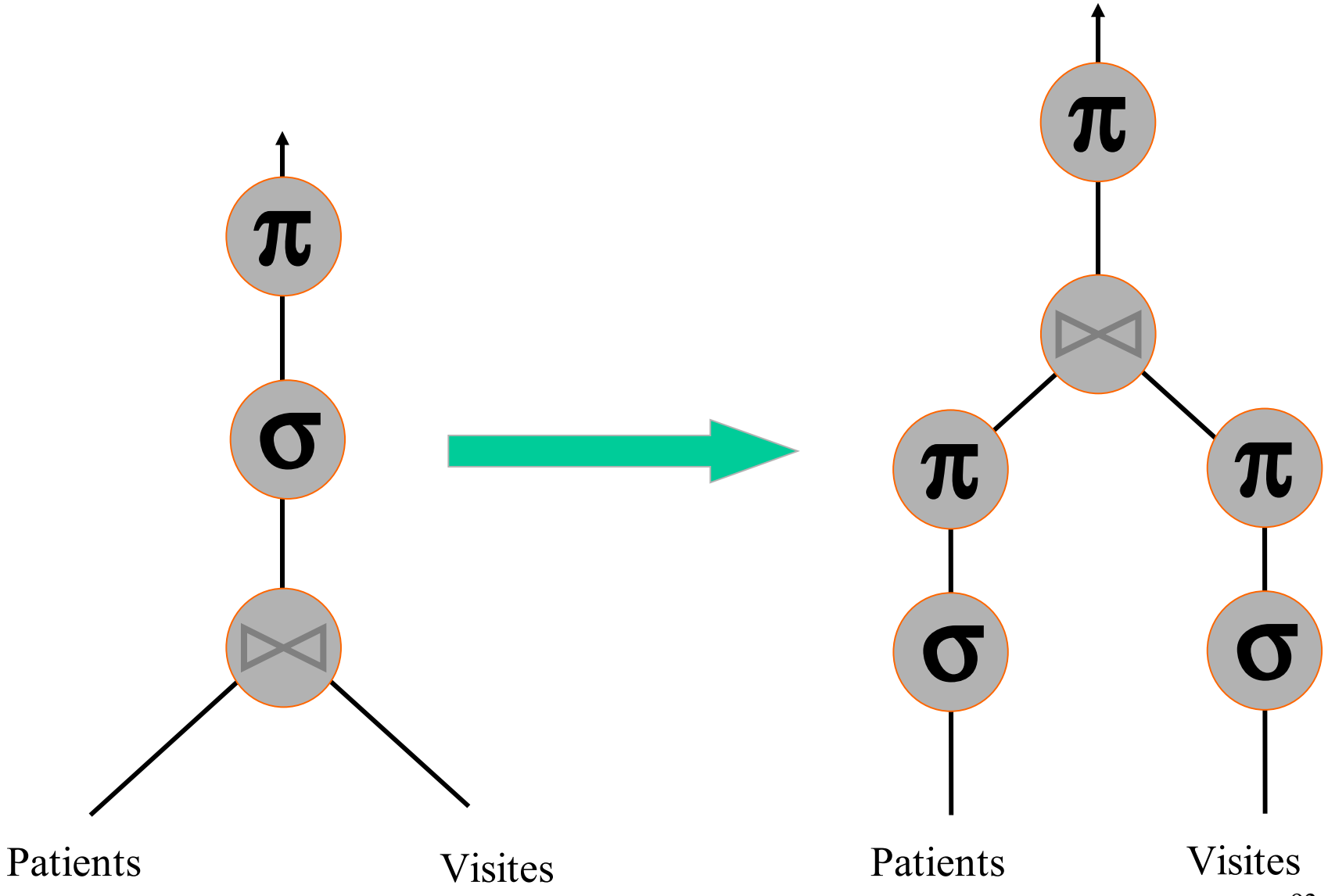


Exemple de plan d'exécution

```
Select   Patients.Nom, Patients.Prénom
From     Patients, Visites
Where    Patients.Id-P = Visites.Id-P
and      Patients.Ville = 'Paris'
and      Visites.Date = '15 juin'
```



Plan d'exécution optimisé



Bases de Données

Nicolas Loménie

LANGAGE SQL

Langage SQL

Origines et Evolutions

- SQL est dérivé de l'algèbre relationnelle et de SEQUEL (1975)
- Il a été intégré à SQL/DS, DB2, puis ORACLE, INGRES, …
- Il existe trois versions normalisées de base, du simple au complexe :
  - –SQL1 86 version minimale
  - –SQL1 89 addendum (intégrité)
  - –SQL2 (92) langage relationnel complet
- Une version SQL3 (99) étendue (objets, règles) est en intégration
- La plupart des systèmes supportent SQL2 complet

Opérations

- Opérations de base
  - –SELECT, INSERT, UPDATE, DELETE

- Opérations additionnelles
  - –définition et modification de schémas
  - –définition de contraintes d'intégrité
  - –définition de vues
  - –accord des autorisations
  - –gestion de transactions

SQL1 - 86

- LANGAGE DE DEFINITIONS DE DONNEES
  - **CREATE TABLE**
  - **CREATE VIEW**

- LANGAGE DE MANIPULATION DE DONNEES
  - **SELECT** - **OPEN**
  - **INSERT** - **FETCH**
  - **UPDATE** - **CLOSE**
  - **DELETE**

- LANGAGE DE CONTROLE DE DONNEES
  - **GRANT** et **REVOKE**
  - **BEGIN** et **END TRANSACTION**
  - **COMMIT** et **ROLLBACK**

Conventions méta-syntaxiques

A::=B se lit *A est redéfini par B*
A|B se lit *A ou B*
[A] se lit *A est optionnel*
A^+ est équivalent à *A[,..,A]*

Les symboles ::=, | , [,], $^+$ ne figurent jamais dans les expressions SQL

SELECT: Forme Générale

- **SELECT** <liste de projection>
  - **FROM** <liste de tables>
  - [**WHERE** <critère de jointure> **AND** <critère de restriction>]
  - [**GROUP BY** <attributs de partitionnement>]
  - [**HAVING** <critère de restriction>]
  - [**ORDER BY** {nom_champ | 1} [**ASC | DESC** [, ..., {nom_champN | N} ASC | DESC }]]]

- Restriction :
  - arithmétique (=, <, >, ≠, ≥,≤)
  - textuelle (LIKE)
  - sur intervalle (BETWEEN)
  - sur liste (ou ensembles issus de SELECT par exemple) (IN)
  -
- Possibilité de blocs SELECT imbriqués par :

  IN, EXISTS, NOT EXISTS, ALL, SOME, ANY

Fonction et Agrégat

- FONCTION et ATTRIBUT CALCULE :
  - –Fonction de calcul en ligne appliquée sur un ou plusieurs attributs
  - –Exemple : **SELECT TVA, HT, (1+TVA) *TTC FROM ACHATS**

- AGREGAT : **GROUP BY et HAVING**
  - –Partitionnement d'une relation selon les valeurs d'un groupe d'attributs, suivi d'un regroupement en ligne des tuples liés par une fonction de calcul en colonne (SUM, MIN, MAX, AVG, COUNT, …)
  - –La clause HAVING permet d'appliquer un critère de restriction sur un GROUPE de tuples !
  - –En opposition à la clause WHERE qui agit sur chaque tuple.

Exemples d'agrégats

VINS	CRU	MILL	DEGRE	QUANTITE
	CHABLIS	1977	10.9	100
	CHABLIS	1987	11.9	250
	VOLNAY	1977	10.8	400
	VOLNAY	1986	11.2	300
	MEDOC	1985	11.2	200

SELECT AVG(DEGRE)
FROM VINS;

AVG	DEGRE
	11.2

SELECT CRU, SUM(QUANTITE)
FROM VINS
GROUP BY CRU;

SUM	CRU	QUANTITE
	CHABLIS	350
	VOLNAY	700
	MEDOC	200

On regroupe sur la colonne CRU mais on fait des statistiques sur la colonne QUANTITE !

Forme générale de la condition de restriction

<**search condition**> ::= [NOT]

<**nom_colonne**> θ (**constante** | <**nom_colonne**>)

<**nom_colonne**> LIKE <**modèle_de_chaîne**>

<**nom_colonne**> IN <**liste_de_valeurs**>

<**nom_colonne**> θ (ALL | ANY | SOME) <**liste_de_valeurs**>

EXISTS <**liste_de_valeurs**>

UNIQUE <**liste_de_valeurs**>

<**tuple**> MATCH [UNIQUE] <**liste_de_tuples**>

<**nom_colonne**> BETWEEN **constante** AND **constante**

<**search condition**> AND | OR <**search condition**>

avec

θ ::= < | = | > | ≥ | ≤ | <>

- Remarque: <**liste_de_valeurs**> **peut être dynamiquement déterminée par une requête**

Soit le schéma de base de données suivant :

VITICULTEURS (NVT, NOM, PRENOM, VILLE, REGION)
VINS (NV, CRU, MILLESIME, DEGRE, PRIX , NVT)
BUVEURS (NB, NOM, PRENOM, VILLE)
ABU (NV, NB,DATE,QTE)

Exemples de Questions (1) et Expression SQL de la Jointure

- Q1: Crus des vins sans doubles.

```
SELECT  DISTINCT CRU
FROM VINS;
```

- Q2: Noms des buveurs ayant bus des Beaujolais 97 ou 98

```
SELECT DISTINCT NOM
FROM BUVEURS B, VINS V, ABU
WHERE B.NB = ABU.NB
AND ABU.NV = V.NV
AND CRU  LIKE '%BEAUJOLAIS%'
AND MILLESIME  IN (1987, 1988);
```

Exemples de Questions (2)

- Q3 : Noms et prénoms des buveurs de vins dont le cru commence par B, de degré inconnu ou compris entre 11 et 13.

```
SELECT NOM, PRENOM
FROM BUVEURS B, VINS V, ABU A
WHERE B.NB = A.NB  AND A.NV = V.NV
AND CRU LIKE "B%"
AND (DEGRE BETWEEN 11 AND 13 OR DEGRE IS NULL);
```

- Q4 : Noms des crus bus par au moins un buveurs.

```
SELECT DISTINCT CRU   /* Le vrai PROJECT*/
FROM VINS V
WHERE EXISTS ( SELECT  *
                FROM BUVEURS B, ABU A
                WHERE B.NB = A.BNB AND A.NV = V.NV );
```

Exemples de Questions (3)

- Q5: Calculer le degré moyen pour chaque cru.

```
SELECT CRU, AVG(DEGRE)
FROM VINS
GROUP BY CRU;
```

- Q6 : Calculer le degré moyen et le degré minimum pour tous les crus de 94 dont le degré minimum est supérieur à 12.

```
SELECT CRU, AVG(DEGRE), MIN(DEGRE)
FROM VINS
WHERE MILLESIME = 1994
GROUP BY CRU
HAVING MIN(DEGRE) > 12;
```

Requêtes imbriquées (1)

- Q7: Donner les crus des vins qui n'ont jamais été bus

```
SELECT CRU
        FROM VINS V
        WHERE V.NV NOT IN (
          SELECT A.NV
          FROM ABU A );
```

```
SELECT CRU
FROM VINS V
WHERE V.NV <> ALL (
          SELECT A.NV
          FROM  ABU A );
```

Requêtes imbriquées (2)

Q8 : Donner le nom des buveurs qui ont bu tous les vins

/* Appel au concept de Division ensembliste : reformuler la question déclarativement : Donner le nom des buveurs pour lesquels il n'existe pas de vins qui n'ait pas été bu (ou encore pour lesquels il n'existe pas d'association ABU) */

```
SELECT NOM
FROM BUVEURS B
WHERE NOT EXISTS (
        SELECT *
        FROM  VINS V
        WHERE NOT EXISTS (
                SELECT *
                FROM ABU A
                WHERE V.NV = A.NV
                AND A.NB = B.NB) );
```

Q8bis : Donner le nom des vins qui ont été bu par tous les buveur ?

Requêtes imbriquées (3)

- Q9: Donner le numéro et le cru des vins bu exactement une fois

```
SELECT NV, CRU
FROM VINS
WHERE NV  MATCH UNIQUE  (
        SELECT NV
        FROM ABU );
```

Requête Union

Q10 :Donner le numéro et le cru des vins bus plus de 100 fois ou bien jamais bus

```
( SELECT V.NV, V.CRU
FROM VINS V, ABU A
WHERE V.NV = A.NV
GROUP BY V.NV
HAVING COUNT(A.NV) > 100 )
UNION
( SELECT NV, CRU
FROM VINS
WHERE NV NOT IN (SELECT NV FROM ABU) );
```

Create Table

- **CREATE TABLE <relation name>**
  (<attribute definition>+)
  [{PRIMARY KEY | UNIQUE} (<attribute name>+)]
- avec :
  <attribute definition> ::= <attribute name> <data type>
  [NOT NULL [{UNIQUE | PRIMARY KEY}]]
- Exemple :

```
CREATE TABLE VINS
  (  NV    INTEGER  PRIMARY KEY
     CRU CHAR VARYING
     MILL  INTEGER NOT NULL,
     DEGRE FIXED 5.2 );
```

Commande INSERT

- **INSERT INTO <relation name>**
  [(attribute [,attribute] …)]
  {VALUES (<value spec.> [, <value spec.>]) …| <query spec.>}

- Exemples

```
INSERT INTO VINS (NV, CRU, MILLESIME)
VALUES (112, (JULIENAS', NULL);

INSERT INTO BUVEURS (NB,NOM,PRENOM)
SELECT  NVT, NOM, PRENOM
FROM VITICULTEURS
WHERE VILLE LIKE '%DIJON%';
```

Commande UPDATE

- **UPDATE <relation name>**
  SET <attribute = {value expression | NULL}
  [<attribute> = {value expression | NULL}] …
  [WHERE <search condition>]

- EXEMPLE

```
UPDATE ABUS
SET QTE = QTE * 1.1
WHERE ABUS.NV IN
    SELECT NV
    FROM VINS
    WHERE CRU = 'VOLNAY' AND MILLESIME = 1990;
```

Commande DELETE

- **DELETE FROM <relation name>**
  [WHERE <search condition>]

- EXEMPLE

```
DELETE FROM ABUS
WHERE NV IN
    SELECT NV
    FROM VINS
    WHERE DEGRE IS NULL
```

SQL1 - 89 : Intégrité

- Valeurs par défaut :

```
CREATE TABLE VINS
( NV INT UNIQUE,
  CRU CHAR(10),
  ANNEE INT,
  DEGRE FIXED (5,2) ,
  NVT INT,
  PRIX  INT DEFAULT 40 );
```

- Contraintes de domaine :

```
SALAIRE INT CHECK BETWEEN 6000 AND
  100000
```

SQL1 - 89 : Contrainte référentielle

- Clé primaire et contrainte référentielle :

```
CREATE TABLE VINS
( NV INT PRIMARY KEY,
  CRU CHAR(10),
  ANNEE INT,
  DEGRE FIXED (5,2) ,
  NVT INT REFERENCES VITICULTEURS,
  PRIX INT DEFAULT 40 );
```

- Référence en principe la clé primaire :
  - celle de VITICULTEURS

SQL1 - 89 : Création de tables

CREATE TABLE
<nom_table>
(<def_colonne>$^{+}$
[<def_contrainte_table>$^{+}$]) ;

< def_colonne > ::=
<nom_colonne> < type | nom_domaine >
[CONSTRAINT nom_contrainte
< NOT NULL | UNIQUE | PRIMARY KEY |
CHECK (condition) | REFERENCES nom_table (liste_colonnes) >]
< def_contrainte_table > ::= CONSTRAINT nom_contrainte
< UNIQUE (liste_colonnes) | PRIMARY KEY (liste_colonnes) |
CHECK (condition) |
FOREIGN KEY (liste_colonnes) REFERENCES nom_table (liste_colonnes) >
[NOT] DEFERRABLE

```
CREATE TABLE EXPEDITIONS
  ( numExp INTEGER PRIMARY KEY
        date_exp DATE,
        qte QUANTITE,
        CONSTRAINT refCom FOREIGN KEY numExp
          REFERENCES  COMMANDES (numCom) DEFERRABLE
) ;
```

•L'association d'un nom à une contrainte est optionnelle.

•Ce nom peut être utilisé pour référencer la contrainte (ex: messages d'erreurs).

•DEFERRABLE : vérification de contrainte différable (à la fin d'une transaction par exemple)

SQL2 (1992)

Trois niveaux distingués :

- Entry SQL2 = SQL89 + manques
- Intermediate SQL2 =Compléments relationnels
- Full SQL2 = Gadgets en plus

Entry SQL2

- Codes réponses SQLSTATE
- Renommage des colonnes résultats
- Mots clés utilisables entre " "
- Méta-base normalisée (schémas)

SQL2 Intermediate : Création de domaines

CREATE DOMAIN <nom> <type> [valeur] [Contrainte]

<type> ::=

CHAR **[(n)]** | VARCHAR **[(n)]**

BIT **[VARYING] [(n)]**

SMALLINT | INTEGER

NUMERIC **[p, [q]]** | DECIMAL **[p, [q]]** | FLOAT **[n]**

DATE | TIME | TIMESTAMP | INTERVAL

<valeur> ::= DEFAULT

constante

USER | NULL

CURRENT_DATE | CURRENT_TIME | CURRENT_TIMESTAMP

Exemples de domaine

- **CREATE DOMAIN cru VARCHAR(20) ;**

- **CREATE DOMAIN couleur_vins CHAR(10) DEFAULT 'rouge' ;**

- **CREATE DOMAIN date_commande DATE DEFAULT CURRENT_DATE ;**

- **CREATE DOMAIN quantite SMALLINT ;**

- Possibilité de créer des domaines avec contraintes :

```
CREATE DOMAINE MONEY IS DECIMAL (5,2)
  DEFAULT (-1)
  CHECK (VALUE = -1 OR VALUE > 0)
  NOT NULL;
```

SQL2 : Types

- Types de données date avec opérations :
  - DATE, TIME et TIMESTAMP
  - Intervalles de temps

- Cascade des mises à jour :

  - Suppression en cas d' intégrité référentielle avec options
    - Cascader les suppressions (CASCADE)
    - Rendre nul l'attribut référençant (NULLIFY)

- Différents alphabets et ordres de lettres

Traitement des chaînes de caractères

expression_caractère ::=

- **'chaîne_caractère'**
- **<nom_colonne>**
- USER
- UPPER (**expression_caractère**)
- LOWER (**expression_caractère**)
- CHARACTER_LENGTH (**expression_caractère**)
- SUBSTRING (**expression_caractère** FROM **début** FOR **longueur**)
- POSITION (**expression_caractère** IN **expression_caractère**)
- CAST (expression AS type | domaine)
- **expression_caractère** | |**expression_caractère**

Exemple de chaînes de caractères

Q11 :Donner le numéro et les 5 premières lettres du cru en majuscules pour chaque vin dont le cru possède plus de 10 lettres

```
SELECT NV, UPPER(SUBSTRING(CRU FROM 1 TO 5)
FROM VINS
WHERE CHARACTER_LENGTH(CRU) > 10
```

Traitement du type date

expression_date_temps ::=

| constante
| <nom_colonne>
| CURRENT_DATE
| CURRENT_TIME [précision]
| CURRENT_TIMESTAMP [précision]
| EXTRACT (champ FROM source)
| CAST (expression AS type | domaine)

Exemple d’utilisation du type date

```
UPDATE VINS
SET ANNEE = EXTRACT (YEAR FROM CURRENT_DATE)
WHERE   NV = 10

UPDATE COMMANDES
SET DATE_COM = CAST ('1996-10-23' AS DATE)
WHERE   NV = 10
```

SQL2 : Opérateurs

- Jointure externe (outer-join) :

```
SELECT …
FROM R1 [NATURAL] [{LEFT | RIGHT}] JOIN R2  [ON (A=B)], …
WHERE …
```

- Expressions de SELECT
  - [OUTER] UNION
  - INTERSECT
  - EXCEPT

Jointure Naturelle

La jointure naturelle est une équijointure dont la condition porte sur l'égalité de valeurs entre tous les attributs de même nom, des relations concernées. Le schéma de la relation résultante correspond à une concaténation de l'ensemble des attributs des deux relations dont elle est issue, autour du ou des attributs communs.

Exemple de Jointure Externe

Q12 :Afficher le numéro et le cru des vins ainsi que la quantité bue pour ceux d'entre eux qui ont été bus

```
SELECT NV, CRU, QTE
FROM VINS NATURAL LEFT JOIN COMMANDES
```

- ou bien

```
SELECT NV, CRU, QTE
FROM VINS LEFT JOIN COMMANDES USING (NV)
```

- ou encore

```
SELECT NV, CRU, QTE
FROM VINS LEFT JOIN COMMANDES ON V.NV = C.NV
```

SQL2 : Création d'index

CREATE [UNIQUE] INDEX [nom_index]
ON nom_table
(<nom_colonne [**ASC** | **DESC**] > *);

Exemple:

CREATE UNIQUE INDEX index_exp

ON EXPEDITION (num_exp **ASC**) ;

Sur quelles colonnes faire porter un index ?
Sur celles qui sont le plus souvent utilisées dans les critères de recherche

Quel index utiliser ?
Arbres B (B-tree) : index par défaut et méthode la plus puissante
Arbres R (R-tree) : index pour les opérations sur les données géométriques
CREATE INDEX geo_ids ON polygones USING RTREE (forme)

PostgreSQL :
->INSERT INTO test values (4, ‘fff’);
->ANALYSE test; \\pour mettre à jour les stat de la table)
->EXPLAIN select * from test where i<34;
->CREATE INDEX testindex on test(i);
->EXPLAIN select * from test where i=34;

SQL2 : Création de schéma

CREATE SCHEMA [nom_schéma]
[**AUTHORIZATION** <nom_utilisateur>]
[**DEFAULT CHARACTER SET** <jeu_car>]
[<éléments_du_schéma> *];

Exemple:

CREATE DATABASE VIGNOBLES // non normalisé

CREATE SCHEMA COOPERATIVE

CREATE DOMAIN ...
CREATE TABLE ...

SQL2 : Modification de schéma

DROP
- **DOMAIN** <nom_domaine>
- **TABLE** <nom_table>
- **INDEX** <nom_index>
- **SCHEMA** <nom_schema>

[**CASCADE** | **RESTRICT**] ;

RESTRICT interdit la destruction d'un objet référencé (via une contrainte d'intégrité, une vue, ...) alors que CASCADE propage la destruction

ALTER TABLE <nom_table>

ADD

COLUMN <def_colonne>

CONSTRAINT <def_contrainte_table >

ALTER <def_colonne>

DROP

COLUMN <nom_colonne>

CONSTRAINT <nom_contrainte >

Exemple :

ALTER TABLE EXPEDITION

ADD COLUMN adresse_livraison **VARCHAR(**30**)**

ALTER quantite **INTEGER NOT NULL** ;

SQL2 Full

- Extension des dates et temps
- Expressions étendues avec correspondances de colonnes (ASSERTION)
- Possibilité de SELECT en argument d'un FROM
- Vues concrètes
- Contraintes d'intégrité multi-tables
- Contrôles d'intégrité différés (DEFERRABLE)

SQL3 ou SQL99 : BD Avancées

- Composants multiples
  - Framework, Foundation, Binding
  - CLI, PSM
  - **Transactions, temporal, real-time**
  - Multimédia

- Interface client
  - SQL/CLI du SAG (X/OPEN)
- Procédures stockées
  - Persistent Stored Modules (PSM)
- Objets
- Requêtes récursives
- Déclencheurs (trigger)

Un Standard depuis 1999

- Proposition concurrente de l'ODMG
  - –Accord entre constructeurs de SGBD Objets
  - –Support du modèle pur objet de l'OMG
  - –Variation de SQL traitant des collections imbriquées
- Accord ANSI X3 H2 et ODMG
  - –Définition d'un langage d'interrogation intégrant relationnel et objet
  - –Convergence relationnel-objet vers SQL3
- SQL3 est le standard depuis 1999
  - –En matière d'objet aussi ...

SQL comprend quatre parties :

- Le langage de définition de schéma (**Tables**, Vues, Droits)
- Le langage de manipulation (**Sélection et mises à jour**)
- La spécification de modules appelables (Procédures)
- L'intégration aux langages de programmation (Curseurs)

Conclusion

- **Un standard de plus en plus complet et de plus en plus suivi** :
  - –Attention aux approximations et imitations incomplètes
  - –Tout ou presque existe dans les propositions SQL2 ou SQL3
  - –Une référence pour implémenter et utiliser chaque aspect des BD

- **Le langage universel sur lequel s'appuie les progiciels**

- http://sqlpro.developpez.com/cours/sqlaz/ddl/?page=sommaire

Bases de Données

Nicolas Loménie

VUES et DROITS

VUES et DROITS

Indépendance Physique : SQL

Indépendance des programmes d'applications vis à vis du modèle physique :

- –Possibilité de modifier les **structures de stockage** (fichiers, index, chemins d'accès, …) sans modifier les programmes;
- –Ecriture des applications par des **non-spécialistes des fichiers** et des structures de stockage;
- –Meilleure **portabilité** des applications et **indépendance** vis à vis du matériel.

Indépendance Logique

Les applications peuvent définir des **vues logiques** de la BD

Gestion des médicaments

Nombre_Médicaments

Id-	Nom	Description	Nombre
1	Aspegic 1000	……	30
2	Fluisédal	……	20
3	Mucomyst	……	230
….	……..	……	…..

Cabinet du Dr. Masse

Visites

Id-D	Id-P	Id-V	Date	Prix
1	2	1	15 juin	250
2	3	4	1 mars	250

Prescription

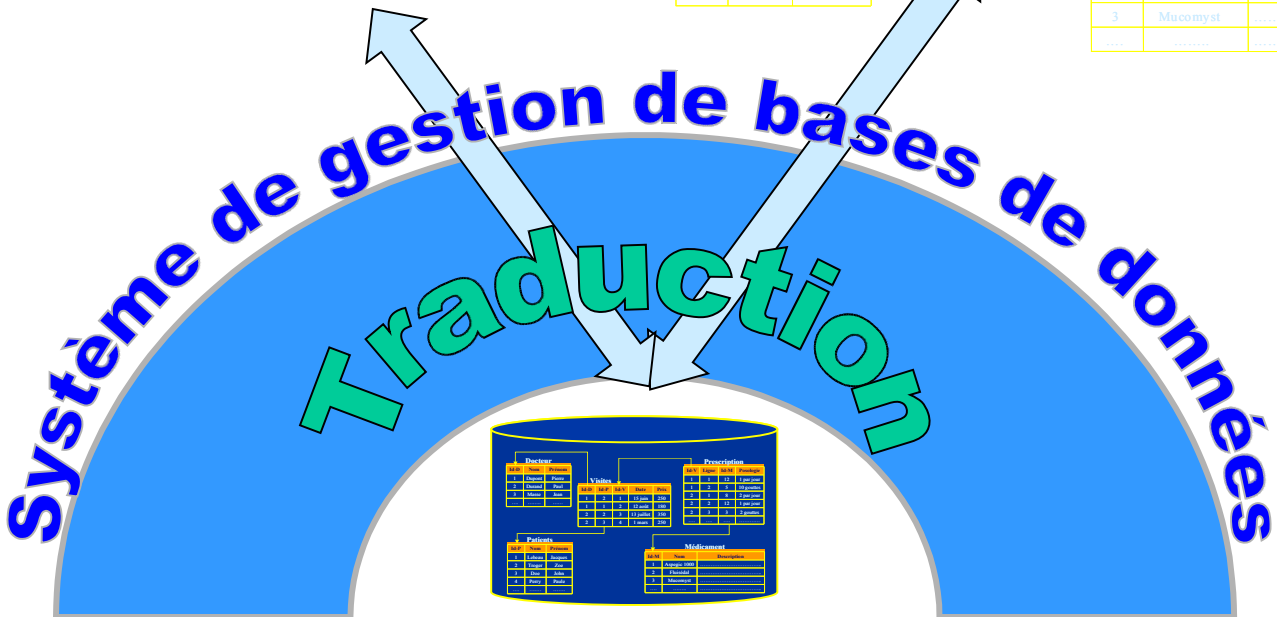
Id-V	Ligne	Id-M	Posologie
1	1	12	1 par jour
1	2	5	10 gouttes
….	….	….	…….

Patients

Id-P	Nom	Prénom
1	Lebeau	Jacques
2	Troger	Zoe
….	…….	…….

Médicament

Id-M	Nom	Description
1	Aspegic 1000	……
2	Fluisédal	……
3	Mucomyst	……
….	…….	……



Avantages de l'indépendance logique

- Possibilité pour chaque application **d'ignorer** les besoins des autres (bien que partageant la même BD).
- Possibilité **d'évolution de la base de données** sans réécriture des applications :
  - ajout de champs, ajout de relation, renommage de champs.
- Possibilité **d'intégrer des applications existantes** sans modifier les autres.
- Possibilité de limiter les conséquences du partage : **Données confidentielles.**

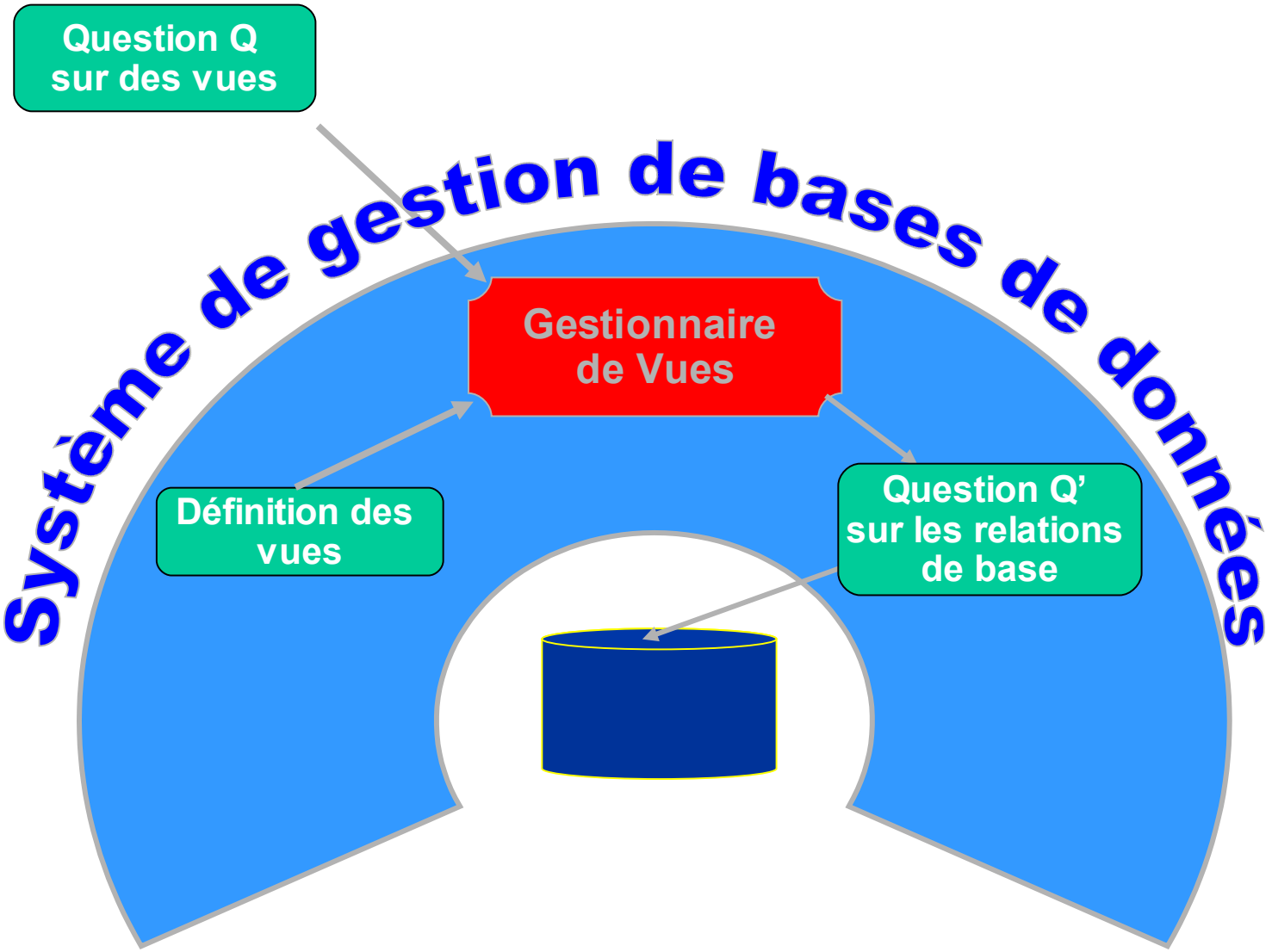
Gestion des vues

- Les vues permettent d'implémenter l'indépendance logique en permettant de créer des **objets virtuels**
- Vue = Question SQL stockée
- Le SGBD stocke la **définition** et non le résultat
- Exemple : la vue des patients parisiens

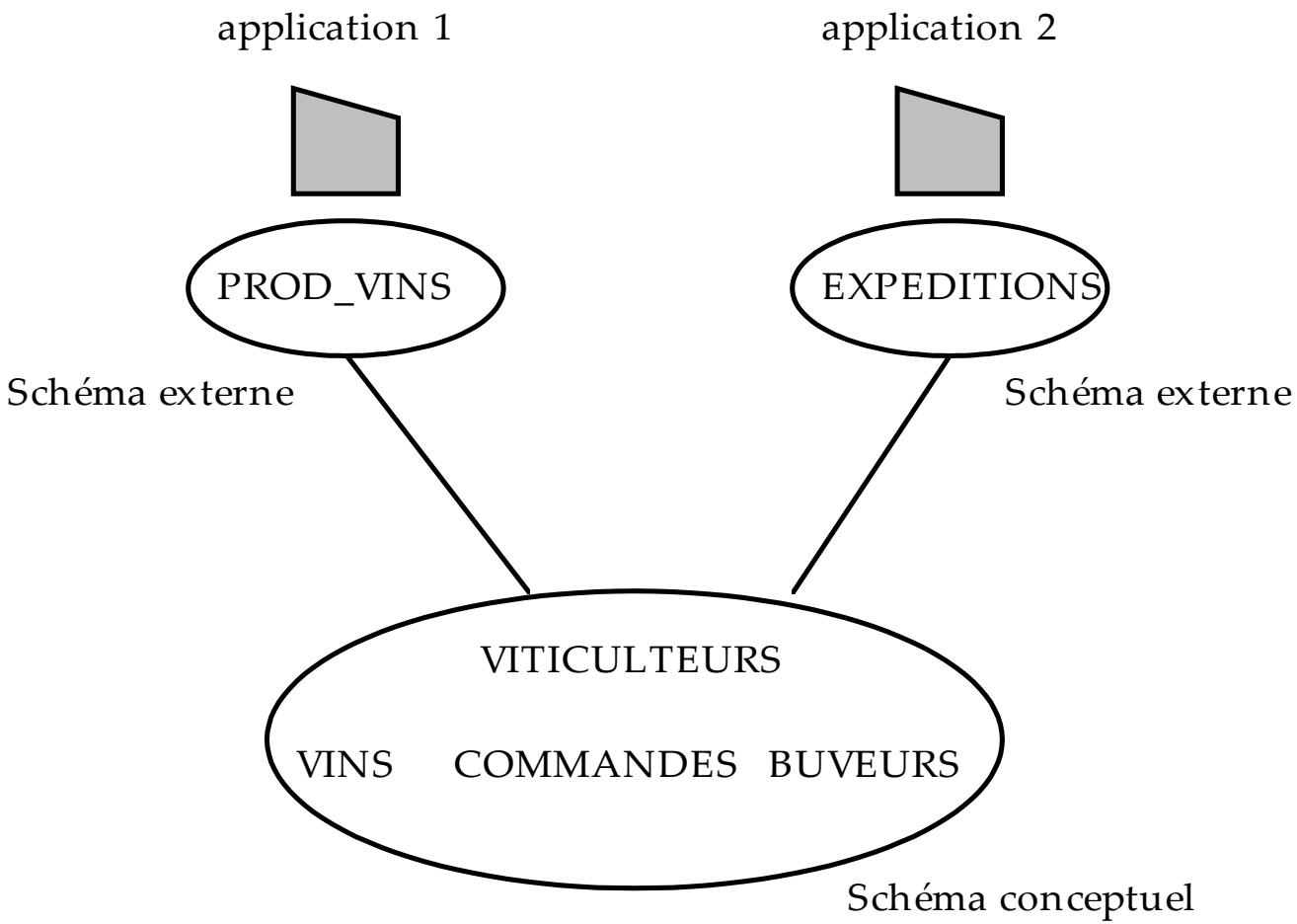
```
Create View Parisiens as (
        Select      Nom, Prénom
        From Patients
        Where       Patients.Ville = 'Paris' )
```

- Relation d'un schéma externe déduite des relations de la base par une question
- Exemple : GrosBuveurs
  - CREATE VIEW GrosBuveurs AS
  - SELECT NB, Nom, Prénom,
  - FROM Buveurs, Abus
  - WHERE Buveurs.NB = Abus.NB and Abus.Quantité > 100
- Calcul de la vue
  - Une vue est une fenêtre dynamique sur la BD et est recalculée à chaque accès.
  - Une vue peut être matérialisée (vue concrète).

Le SGBD **transforme** la question sur les vues en question sur les relations de base



Les vues externes



- Schéma conceptuel :

  VINS (NV, CRU, ANNEE, DEGRE, NVIT)

  VITICULTEURS (NVIT, NOM, PRENOM, VILLE)

  BUVEURS (NB, NOM, PRENOM, VILLE)

  COMMANDE (NB, NV, QTE, DATE)

- On peut définir de nombreuses applications particulières qui n'utilisent qu'une partie des données du schéma.

Exemple

APPLICATION = INFLUENCE GEOGRAPHIQUE
SUR LA CONSOMMATION DE L'ANNEE 1996

- **VUES = RESTRICTION-PROJECTION DES RELATIONS DE BASE**

  BUVEURS-1 (NB, VILLE)
  ACHETE-1 (NB, NV, QTE, DATE) // restreint aux dates de l'année 96
  VINS-1 (NV, CRU)

- **VUE = RESTRUCTURATION DU SCHEMA CONCEPTUEL**

  ACHAT-2 (NB, VILLE, CRU, QTE, DATE)

- **VUE = RESTRUCTURATION et AGREGATION**

  CONSOMMATION-PAR-VILLE
  CPV (VILLE, CRU, QTE)

Définition SQL des vues

CREATE VIEW <nom_vue> [(liste_attributs)]
AS <expression_de_sélection>
[WITH CHECK OPTION]

- L'expression de sélection peut porter sur des tables de base et/ou des vues
- Dans le cas de vues modifiables, la clause WITH CHECK OPTION garantit que les tuples insérés (ou modifiés) dans la vue vérifient bien le critère de la vue

Interrogation des vues

MODIFICATION DE QUESTION PAR RESTRUCTURATION SYNTAXIQUE

EXEMPLE :

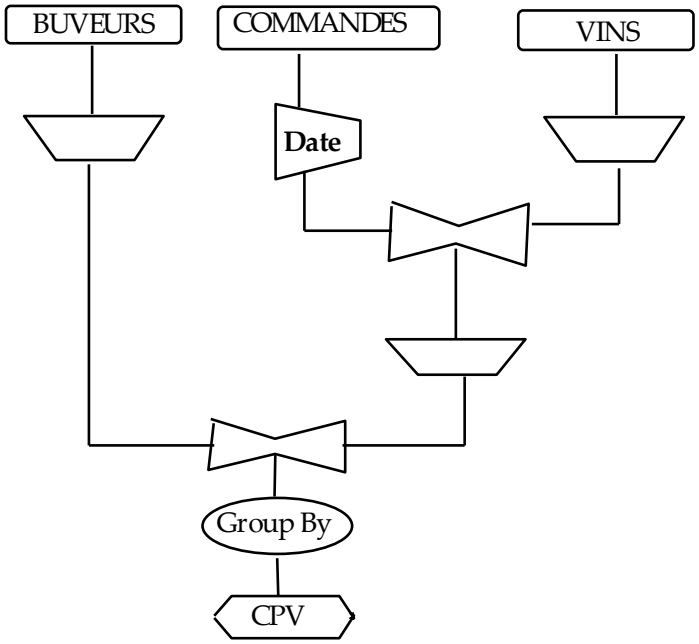
```
SELECT  CRU, QTE
FROM    CPV
WHERE   VILLE = "PARIS"
```

DEVIENT :

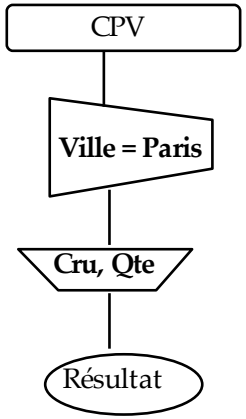
```
SELECT  V.CRU, SUM (C.QTE)
FROM    BUVEURS B, COMMANDES C, VINS V
WHERE   B.NB = C.NB AND
        C.NV = V.NV AND
        C.DATE BETWEEN  '01.01.96' AND '31.12.96' AND
        B.VILLE = 'PARIS'
GROUP BY VILLE, CRU
```

MODIFICATION DE QUESTION PAR RESTRUCTURATION D'ARBRES ALGEBRIQUES

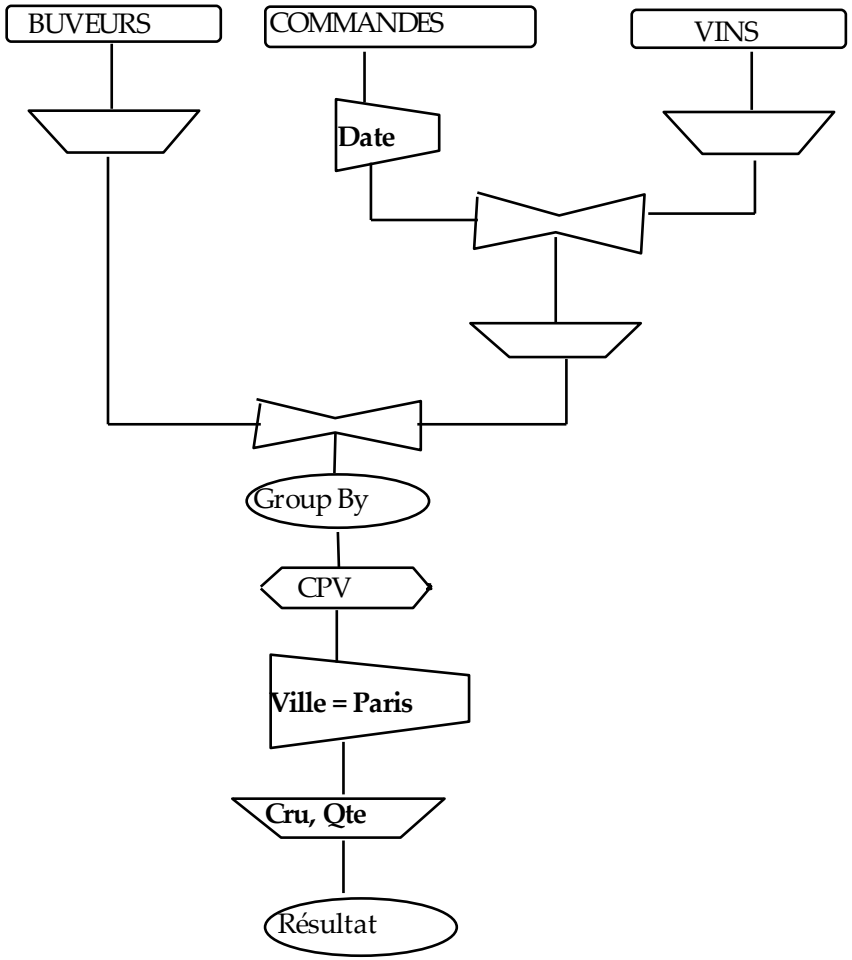
Arbre algébrique de la vue CPV



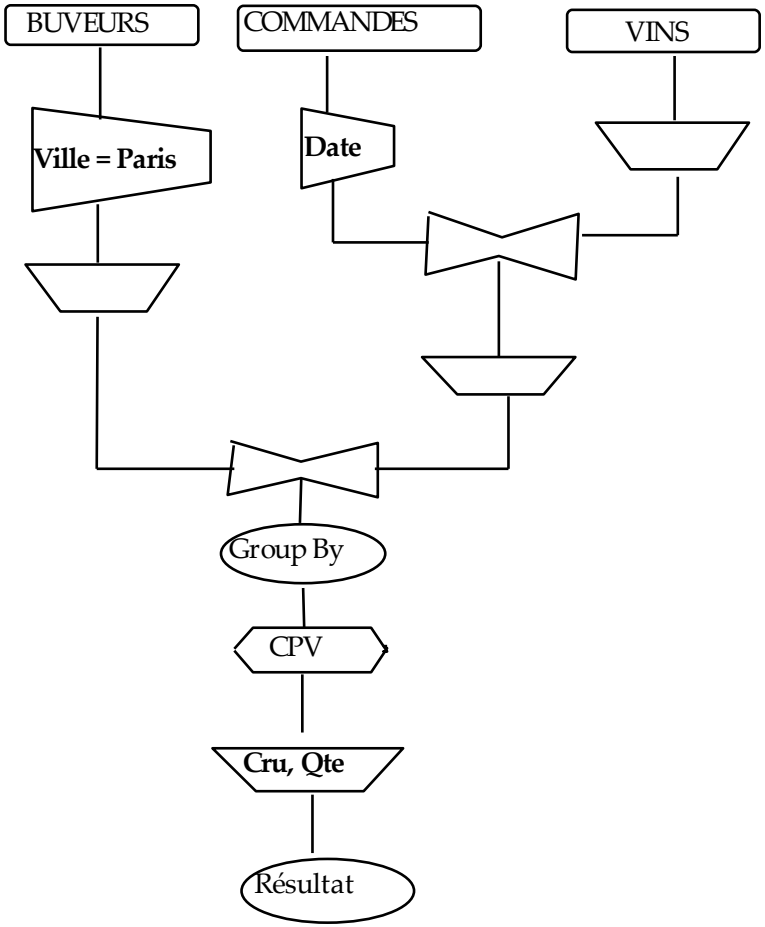
Arbre algébrique de la requête



Les arbres sont mis bout à bout



Puis l'arbre résultat est optimisé



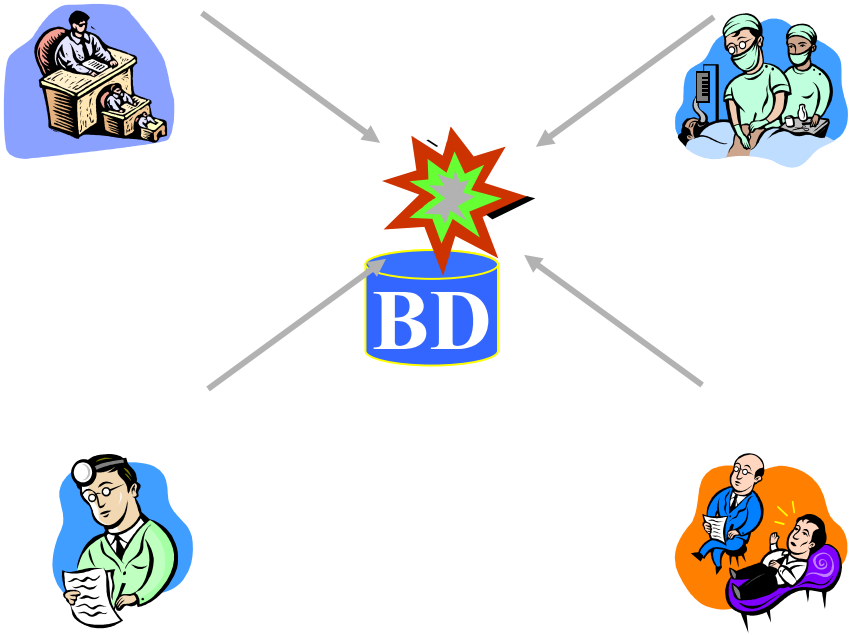
Mises à jour au travers des vues

- Les mises à jour au travers des vues sont rarement définies
  - Ex: Comment reporter une mise à jour sur CPV dans les relations de base ?
- Pour rendre le report de mises à jour possible, la définition de la vue doit respecter certaines contraintes:
  - la clause SELECT doit conserver les clés de toutes les relations de base
  - la clause SELECT ne doit pas contenir de calcul d'agrégat
  - la définition ne doit pas contenir de clause GROUP BY ni HAVING
- Dans SQL2, en plus des conditions précédentes, seules les vue définies à partir d'une relation unique seront considérées comme modifiables

Vues concrètes

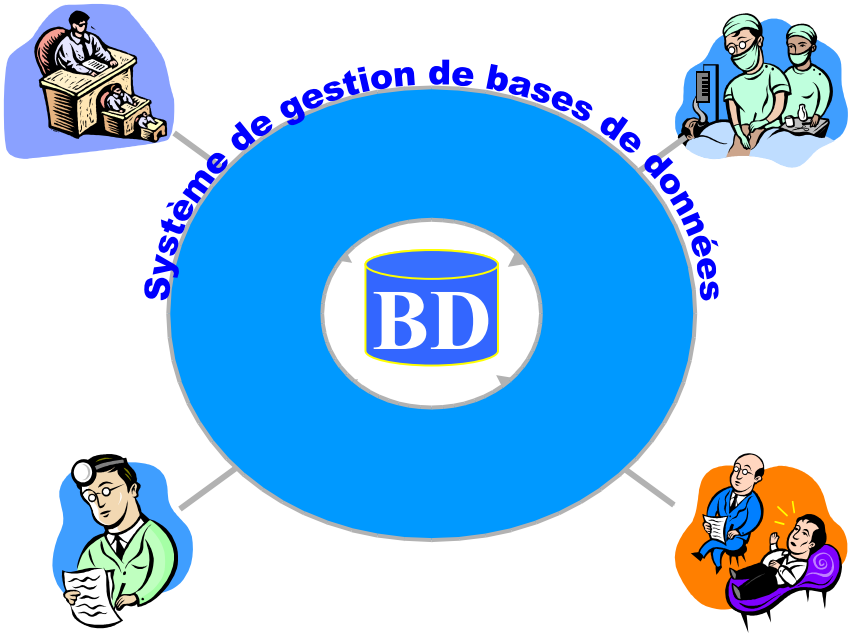
- Vue concrète :
  - Vue dont on a demande l'implantation dans la base de données (virtuel ---> réel)
- Intérêts :
  - interrogations fréquentes,
  - systèmes répartis
- Problèmes :
  - Maj des données de base à répercuter sur la vue
- L'objectif est d'éviter de réévaluer complètement la vue à chaque mise a jour

Partage des données



- Accès concurrent aux mêmes données

➔Conflits d'accès !!



- Le SGBD gère les accès concurrents

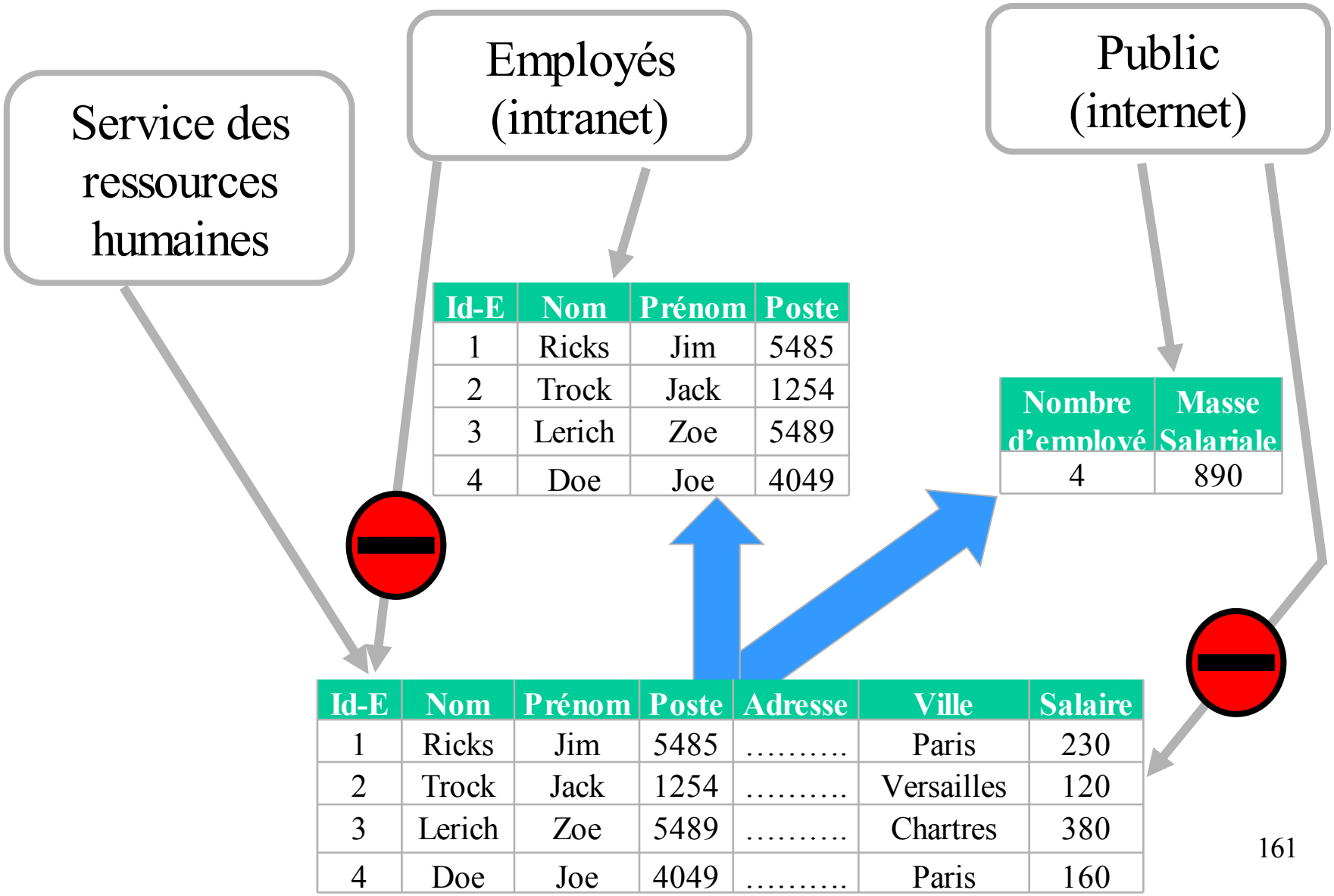
➔ Chacun à l'*impression* d'être seul (Isolation)

➔ Cohérence conservée (Verrouillage)

Confidentialité et Contrôle d'accès aux données

- **Objectif : Protéger les données de la BD contre des accès non autorisés**
- Deux niveaux :
  - Connexion restreinte aux **usagers répertoriés** (mot de passe)
  - **Privilèges** d'accès aux objets de la base
- Usagers : Usager ou groupe d'usagers
- Objets : Relation, **Vue**, autres objets (procédures, triggers, etc.)

Puissance des droits SGBD



Hiérarchie des privilèges

Administrateur Système {users login, …}

Administrateur de la Base {connexion utilisateurs à BD, gestion des schémas}

Propriétaires d'Objets {créateurs d'objets, distribution des droits}

Autres (PUBLIC) {droits obtenus par GRANT}

Définition des droits

GRANT <droits> **ON** <objet>
TO <usagers>
[WITH GRANT OPTION]

<droits> ::= ALL | SELECT | DELETE |
INSERT [<attribut>]| UPDATE [<attribut>]|
REFERENCE [<attribut>]

<objet> ::= TABLE <relation> * | VIEW <vue> *

<usagers> : := PUBLIC| <username> *

Exemple:

GRANT SELECT , UPDATE (DEGRE) **ON TABLE** VINS
TO DUPONT

Suppression des droits

REVOKE <droits> **ON** <objet>
FROM <usagers>

Exemple:

REVOKE ALL **ON TABLE** VINS
FROM DUPONT

Règles d'octroi et de suppression des droits

- On ne peut transmettre que les droits que l'on possède ou qui nous ont été transmis avec “grant option”
- On ne peut supprimer que les droits que l'on a transmis
- La révocation des droits est récursive
- Si un utilisateur U1 a reçu le droit D de la part de plusieurs utilisateurs (U2, U3, ...), il ne perd ce droit que si tous les utilisateurs lui retirent

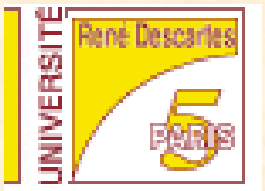
Ces règles garantissent la cohérence des retraits

Autorisations sur une vue

- Une vue est une relation virtuelle définie à partir des relations de base par une expression SQL
- Exemple :

```
CREATE VIEW  GROS_BUVEURS (NB, NOM, QTE_BUE)
AS  SELECT B.NB, B. NOM, SUM(A.QTE)
                FROM BUVEURS B, ABUS A
        WHERE B.NB = A.NB
                GROUP BY NB
```

- AUTORISATIONS SUR LES VUES:
  - Permet un granule d'autorisation très fin
  - Cependant, les mises à jour au travers des vues sont limitées



Bases de Données

Nicolas Loménie

TRANSACTIONS et INTEGRITE

TRANSACTIONS et INTEGRITE

Intégrité Logique

- **Objectif : Détecter les mises à jour erronées**

- Contrôle sur les données élémentaires
  - Contrôle de types: ex: Nom alphabétique
  - Contrôle de valeurs: ex: Salaire mensuel entre 5 et 50kf

- Contrôle sur les relations entre les données
  - Relations entre données élémentaires:
    - Prix de vente > Prix d'achat
  - Relations entre objets:
    - Un électeur doit être inscrit sur une seule liste électorale

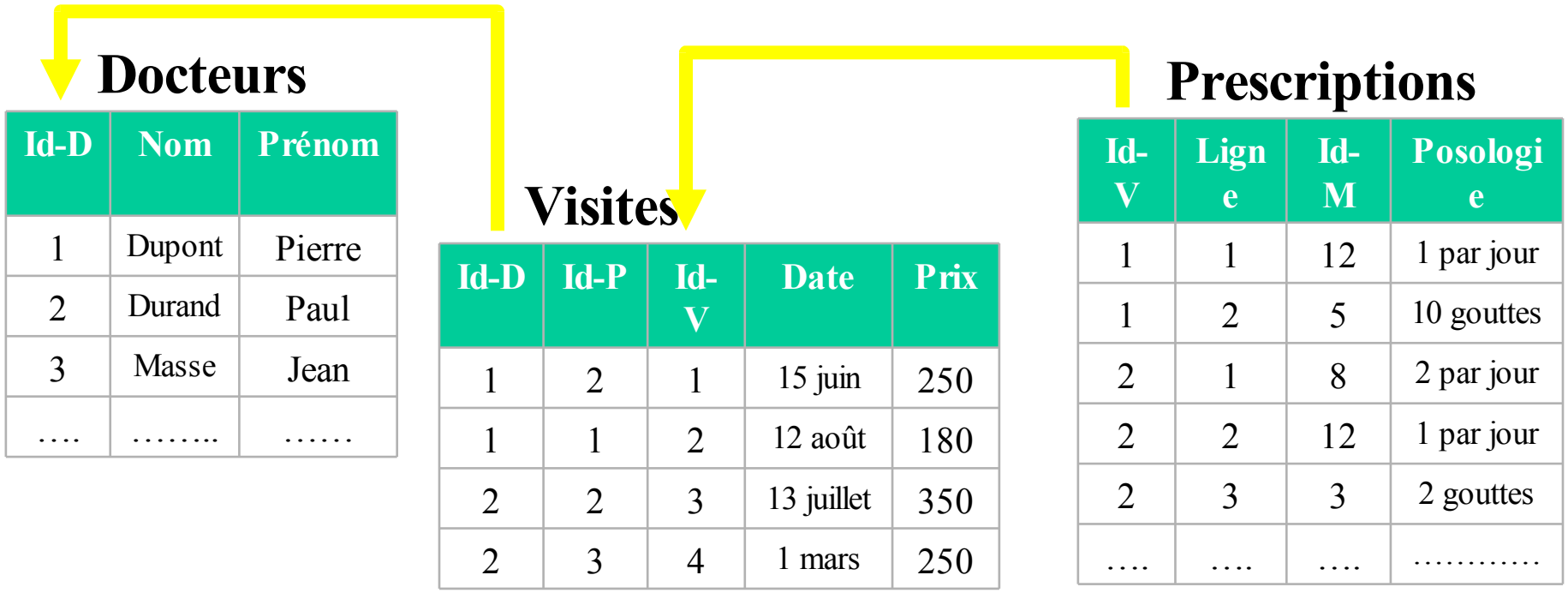
Contraintes d'intégrité

- **Avantages** :
  - **simplification** du code des applications
  - **sécurité renforcée** par l'automatisation
  - **mise en commun** des contraintes

- **Nécessite :**
  - un langage de définition de contraintes d'intégrité
  - la vérification **automatique** de ces contraintes

Exemples de contrainte

Contraintes d'intégrité référentielles



Docteurs

Id-D	Nom	Prénom
1	Dupont	Pierre
2	Durand	Paul
3	Masse	Jean
….	……..	……

Visites

Id-D	Id-P	Id-V	Date	Prix
1	2	1	15 juin	250
1	1	2	12 août	180
2	2	3	13 juillet	350
2	3	4	1 mars	250

Prescriptions

Id-V	Ligne	Id-M	Posologie
1	1	12	1 par jour
1	2	5	10 gouttes
2	1	8	2 par jour
2	2	12	1 par jour
2	3	3	2 gouttes
….	….	….	……

Association des contraintes

- Une contrainte d'intégrité peut être :
  - Associée à un domaine
    - Spécifiée au travers de la clause CREATE DOMAIN
  - Associée à une relation
    - Spécifiée au travers de la clause CREATE TABLE
  - Dissociée
    - Spécifiée au travers de la clause CREATE ASSERTION

Contraintes associées aux domaines

CREATE DOMAIN <nom> <type> [valeur]
[**CONSTRAINT** nom_contrainte **CHECK**
(condition)]

Exemple:

CREATE DOMAIN couleur_vins **CHAR(5) DEFAULT** 'rouge'

CONSTRAINT couleurs_possibles **CHECK**

(VALUE IN ('rouge', 'blanc', 'rosé'))

Contraintes associées aux relations

CREATE TABLE <nom_table>
(<def_colonne> *
[<def_contrainte_table>*]) ;

< **def_colonne** > ::= <**nom_colonne**> < **type** | **nom_domaine** >
[CONSTRAINT **nom_contrainte**
< NOT NULL | UNIQUE | PRIMARY KEY |
CHECK (**condition**) | REFERENCES **nom_table** (**colonne**) >]
[NOT] DEFERRABLE

< **def_contrainte_table** > ::= CONSTRAINT **nom_contrainte**
< UNIQUE (**liste_colonnes**) | PRIMARY KEY (**liste_colonnes**) |
CHECK (**condition**) |
FOREIGN KEY (**liste_colonnes**) REFERENCES **nom_table** (**liste_colonnes**) >
[NOT] DEFERRABLE

Contraintes associées aux relations

```
CREATE TABLE VINS
 (          NV INTEGER PRIMARY KEY,
            couleur COULEURS_VINS,
            cru VARCHAR(20),
            millesime DATE,
            degre CHECK (degre BETWEEN 8 AND 15) NOT DEFERRABLE,
            quantite INTEGER,

            CONSTRAINT dependance_fonctionnelle
                        CHECK (NOT EXISTS (SELECT *
                                           FROM VINS
                                           GROUP BY cru,millesime
                                           HAVING COUNT(degre) > 1)
            NOT DEFERRABLE) ;
```

Contraintes référentielles

```
FOREIGN KEY (liste_colonnes)
REFERENCES nom_table (liste_colonnes)
[ON DELETE {CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {CASCADE | SET DEFAULT | SET NULL}]
[NOT] DEFERRABLE
```

- **Les contraintes référentielles caractérisent toutes les associations**
- **Problème des contraintes référentielles croisées ==> mode DEFERRABLE**
- **En cas de violation de la contrainte, la mise à jour peut être rejetée ou bien une action de correction est déclenchée ==>**

  - **ON DELETE spécifie l'action à effectuer en cas de suppression d'un tuple référencé**
  - **ON UPDATE spécifie l'action à effectuer en cas de mise à jour de la clé d'un tuple référencé**

```
CREATE TABLE ABUS
 ( NB INTEGER NOT NULL,
        NV INTEGER NOT NULL,
        date DATE,
        qte QUANTITE,
        UNIQUE (NB, NV, date)

        CONSTRAINT référence_buveurs
                FOREIGN KEY NB
                REFERENCES  BUVEURS (NB)
                ON DELETE CASCADE
                DEFERRABLE
  ) ;
```

Contraintes dissociées

```
CREATE ASSERTION nom_contrainte CHECK
    (condition)
```

Remarque: les contraintes dissociées peuvent être multi-tables

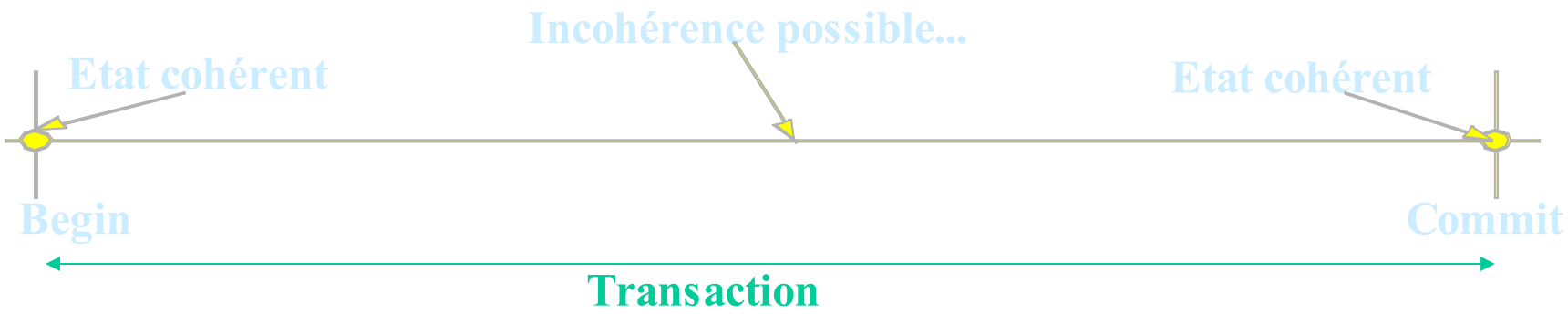
Exemple:

```
CREATE ASSERTION quantite_produite
CHECK    ( (SELECT SUM(quantite) FROM VINS) >
                ( SELECT SUM(quantite) FROM ABUS) )
```

Intégrité Physique

- **Motivations : Tolérance aux fautes**
  - Transaction Failure : Contraintes d'intégrité, Annulation
  - System Failure : Panne de courant, Crash serveur ...
  - Media Failure : Perte du disque
  - Communication Failure : Défaillance du réseau
- **Objectifs :**
  - Assurer l'atomicité des transactions
  - Garantir la durabilité des effets des transactions commises
- **Moyens :**
  - Journalisation : Mémorisation des états successifs des données
  - Mécanismes de reprise

Transaction



Begin T1

CCourant = CCourant - 2000

CEpargne = CEpargne - 3000

CCourant = CCourant + 3000

Commit T1

Faire le lien avec la clause DEFERRABLE : contrainte CCourant > 0

Atomicité et Durabilité

ATOMICITE

Begin
CEpargne = CEpargne - 3000
CCourant = CCourant + 3000
Commit T1

Panne

➔ **Annuler le débit !!**

DURABILITE

Begin
CEpargne = CEpargne - 3000
CCourant = CCourant + 3000
Commit T1

Crash disque

➔ **S'assurer que le virement a été fait !**

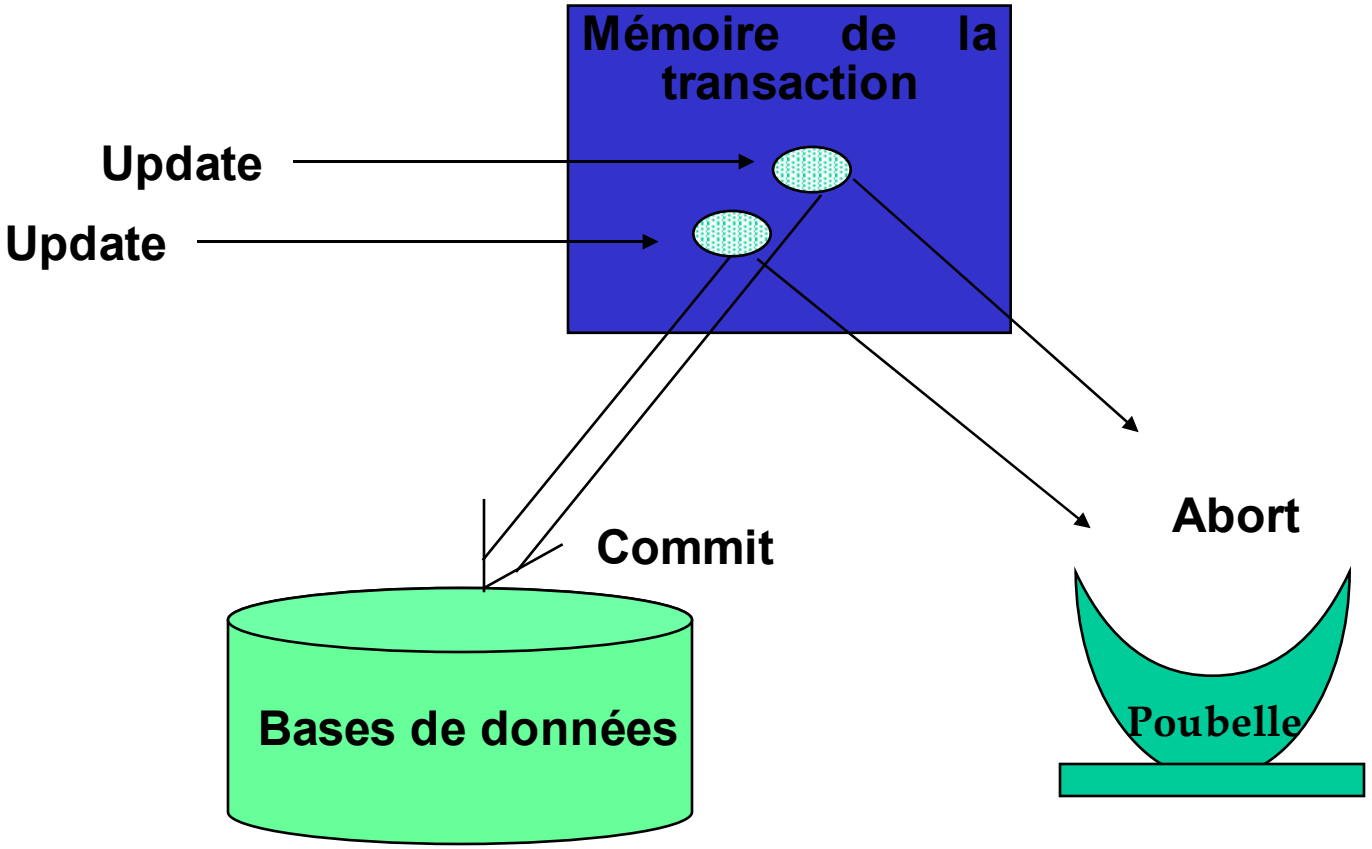
Propriétés des transactions

- Atomicité
  - Unité de cohérence : toutes les mises à jour doivent être effectuées ou aucune.
- Cohérence
  - La transaction doit faire passer la base de donnée d'un état cohérent à un autre.
- Isolation
  - Les résultats d'une transaction ne sont visibles aux autres transactions qu'une fois la transaction validée.
- Durabilité
  - Les modifications d ’une transaction validée ne seront jamais perdue

Commit et Abort

- INTRODUCTION D'ACTIONS ATOMIQUES
  - Commit (fin avec succes) et Abort (fin avec echec)
  - Ces actions s'effectuent en fin de transaction
- COMMIT
  - Validation de la transaction
  - Rend effectives toutes les mises à jour de la transaction
- ABORT
  - Annulation de la transaction
  - Défait toutes les mises à jour de la transaction

Effet logique



Interface applicative

- API pour transaction simple
  - Trid Begin (context*)
  - Commit ()
  - Abort()
- Possibilité de points de sauvegarde :
  - Savepoint Save()
  - Rollback (savepoint) // savepoint = 0 ==> Abort
- Quelques interfaces supplémentaires
  - ChainWork (context*) //Commit + Begin
  - Trid Mytrid()
  - Status(Trid) // Active, Aborting, Committing, Aborted, Committed

- Le tout relié à des journaux de sauvegarde et de reprise -> apprentissage métier

JAVA

```
import java.sql.*;

public class TransactionPairs {

        public static void main(String args[]) {

                String url = "jdbc:mySubprotocol:myDataSource";
                Connection con = null;
                Statement stmt;
                PreparedStatement updateSales;
                PreparedStatement updateTotal;
                String updateString = "update COFFEES " +
                                                        "set SALES = ? where COF_NAME like ?";

                String updateStatement = "update COFFEES " +
                                        "set TOTAL = TOTAL + ? where COF_NAME like ?";
                String query = "select COF_NAME, SALES, TOTAL from COFFEES";

                try {
                        Class.forName("myDriver.ClassName");

                } catch(java.lang.ClassNotFoundException e) {
                        System.err.print("ClassNotFoundException: ");
                        System.err.println(e.getMessage());
                }
```

```
try {

                        con = DriverManager.getConnection(url, "myLogin", "myPassword");

                        updateSales = con.prepareStatement(updateString);
                        updateTotal = con.prepareStatement(updateStatement);
                        int [] salesForWeek = {175, 150, 60, 155, 90};
                        String [] coffees = {"Colombian", "French_Roast",
                                "Espresso", "Colombian_Decaf","French_Roast_Decaf"};
                        int len = coffees.length;
                        con.setAutoCommit(false);
                        for (int i = 0; i < len; i++) {
                                updateSales.setInt(1, salesForWeek[i]);
                                updateSales.setString(2, coffees[i]);
                                updateSales.executeUpdate();

                                updateTotal.setInt(1, salesForWeek[i]);
                                updateTotal.setString(2, coffees[i]);
                                updateTotal.executeUpdate();
                                con.commit();
                        }

                        con.setAutoCommit(true);
                        updateSales.close();
                        updateTotal.close();
                        stmt = con.createStatement();
                        ResultSet rs = stmt.executeQuery(query);

                        while (rs.next()) {
                                String c = rs.getString("COF_NAME");
                                int s = rs.getInt("SALES");
                                int t = rs.getInt("TOTAL");
                                System.out.println(c + "     " +  s + "    " + t);
                        }
                        stmt.close();
                        con.close();

}
```

```
catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                        if (con != null) {
                                try {
                                        System.err.print("Transaction is being ");
                                        System.err.println("rolled back");
                                        con.rollback();
                                } catch(SQLException excep) {
                                        System.err.print("SQLException: ");
                                        System.err.println(excep.getMessage());
                                }
                        }
                }
        }
}
```

Déclencheurs (Triggers)

- Déclencheur :
  - action ou ensemble d'actions déclenchée(s) automatiquement lorsqu'une condition se trouve satisfaite après l'apparition d'un événement
- Un déclencheur est une règle ECA
  - Evénement = mise à jour d'une relation, une opération sur une table (début ou fin), un événement externe (heure, appel,etc.)
  - Condition = optionnelle, équivaut à une clause <WHERE>
  - Action = exécution de code spécifique (requête SQL de mise à jour,exécution d'une procédure stockée, abandon d'une transaction, ...)
- De multiples usages sont possibles :
  - contrôle de l'intégrité
  - maintien de statistiques
  - mise à jour de copies multiples, ...
- **Exemple** : décrémenter un stock à chaque commande du produit

Définition des triggers

CREATE TRIGGER <nom-trigger>
<événement>
[<condition>]
<action >

<événement> ::=
BEFORE | AFTER
{INSERT | DELETE | UPDATE [OF <liste_colonnes>**]}**
ON <nom_de_table>
<condition> ::=
[REFERENCING OLD AS <nom_tuple> **NEW AS** <nom_tuple>**]**
WHEN <condition_SQL**>**
<action> ::=
{requête_SQL **[FOR EACH ROW]**
| exec_procédure **| COMMIT | ROLLBACK**}

Exemples de trigger

```
CREATE TRIGGER degré_croissant
BEFORE UPDATE OF degre ON VINS
REFERENCING OLD AS old_vin NEW AS new_vin
WHEN (new_vin.degre < old_vin.degre)
ROLLBACK
FOR EACH ROW
```

```
CREATE TRIGGER référence_vins
BEFORE DELETE ON VINS
DELETE FROM ABUS
WHERE ABUS.NV = VINS.NV
FOR EACH ROW
```

Equivalent ON DELETE CASCADE sur la table VINS ?

PostgreSQL : CREATE TRIGGER {BEFORE|AFTER} {évènement }
ON nom_table
FOR EACH {ROW|STATEMENT}
EXECUTE PROCEDURE nom_fonction (paramètres)

La fonction *nom_fonction* est développée en PL/pgSQL par exemple

Bilan transactions

- Des techniques complexes
- Un problème bien maîtrisé dans les SGBDR
- La concurrence complique la gestion de transactions
- Les transactions longues restent problématiques
- Enjeu essentiel pour le commerce électronique

- Toutes ces vérifications de contraintes se font par programmation : TRIGGER associé à des fonctions écrites en PL/pgSQL pour postresql, dans le code Java directement par exemple

CONCLUSION

- Le modèle relationnel offre
  - des contraintes d'intégrités riches
  - des droits d'accès intégrés
  - un concept de vue très puissant
  - des methodes d'interrogation simples et efficaces
- Problèmes difficiles :
  - maj au travers des vues
  - vues concrètes

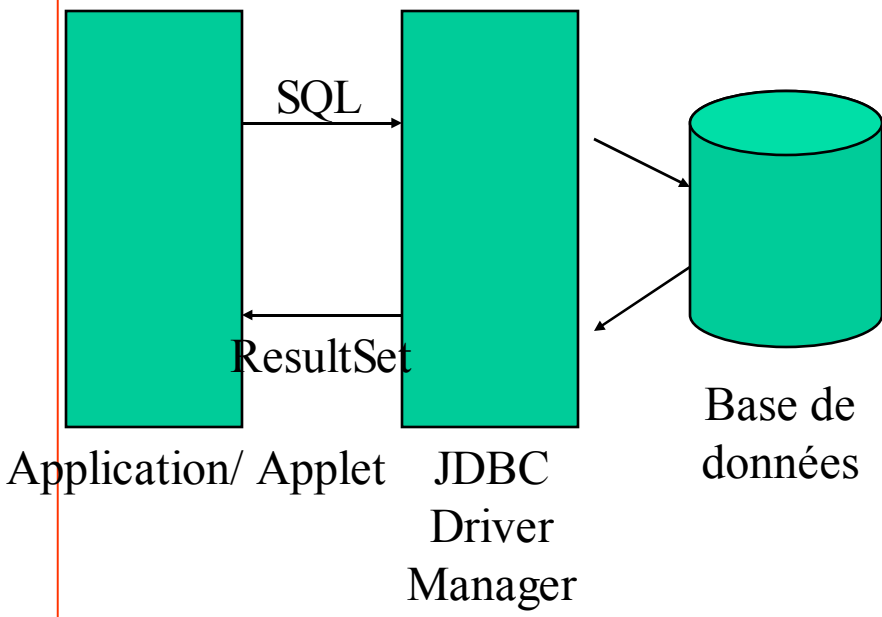
Interfaçage BD/Applications

- C + curseurs
- Java + JDBC

•JDBC : Java Database Connectivity est une API (Interface Applicative) calquée sur ODBC (Object Database Connectivity)

•Classes du package java.sql (SQL3 ++)
•Exportations des interfaces des classes par le JDBC Driver Manager

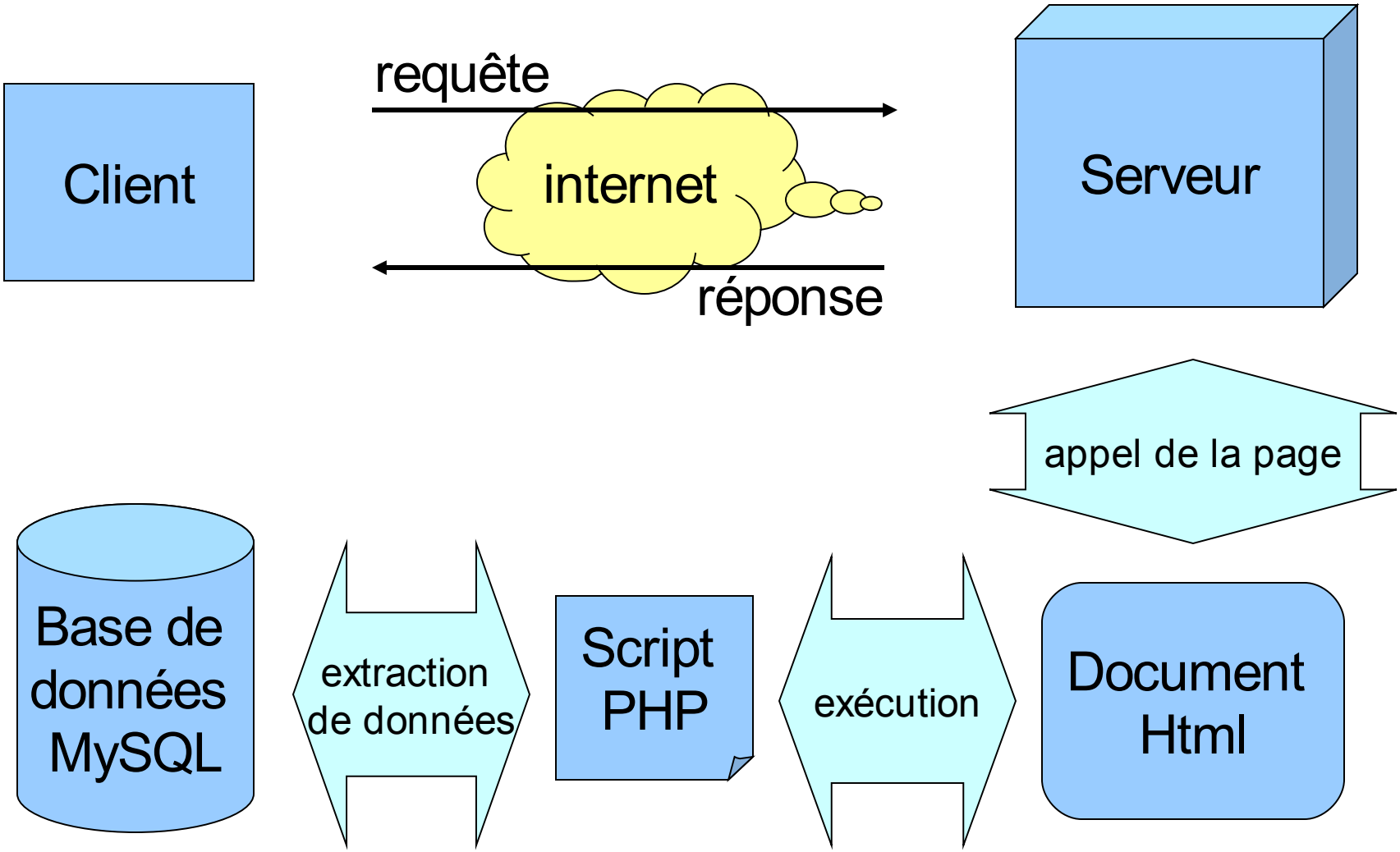
```
// Connexion
Connection con = DriverManager.getConnection(url);
// Création d'une instruction
Statement statement = con.createStatement();
// Exécution d'une requete
String query = "SELECT * FROM Employés";
ResultSet resultset = statement.executeQuery(query);
// Traitement des résultats
while(resultset.next()) {
System.out.println(resultset.getString(2) + " " +
      resultset.getString(3));  }
// fermeture de la connexion
statement.close();
con.close();
```



url = “jdbc.odbc.mabase”, “nombase”,”password”

Interfaçage BD/Web

- php/mysql ou postgresl
- Tomcat/Java/Servlet
- XML


Client
requête
internet
réponse
Serveur
appel de la page
Base de données MySQL
extraction de données
Script PHP
exécution
Document Html

```
<HTML>
<HEAD>
<TITLE>HOPITAL ACCUEIL</TITLE>
</HEAD>
<BODY LINK="#0000ff" BGCOLOR="White">
<CENTER>
<P>
<?php
$link=mysql_connect("diamant","login", $Password) or die ("impossible de se connecter");
mysql_select_db ("login") or die ("Impossible d'acceder a la base de donnees");
$query = "SELECT * FROM hopital";
$result = mysql_query($query) or die ("La requete a echoue");
?>
<table border="1">
<tr>
<td>Nom</td>
<td>Arrondissement</td>
</tr>
<?php
while ($line = mysql_fetch_array($result)){
print "\t <tr>\n";
echo "\t\t<td>".$line["nom_hopital"]."</td>\n";
echo "\t\t<td>".$line["arrondissement"]."</td>\n";
print "\t</tr>\n";
}
print "</table>\n";
mysql_close($link);
?>
<P>
</BODY>
</HTML>
```

Bibliographie :

Les cahiers du programmeur, PostgreSQL, Services Web avec PostreSQL et PHP/XML, Stephane Mariel, Edition Eyrolles

PostgreSQL par la pratique, John C. Worsley, Ed. O'Reilly

postgreSQL

Types et fonctions spécifiques
Très utiles…

Calcul de distances et géomarketing

Si le champ position est de type point, on peut chercher les partenaires proches d'un client :

Catégorie	Type de données	Description	Normalisation
Types date et heure	`date`	Date du calendrier (jour, mois et année).	SQL92
	`time`	Heure.	SQL92
	`time with time zone`	Heure avec les informations sur la zone horaire.	SQL92
	`timestamp with time zone`	Date et heure.	SQL92
	`interval`	Délai quelconque.	SQL92
Types géometriques	`box`	Rectangle à deux dimensions.	Spécifique à PostgreSQL.
	`line`	Ligne infinie dans un plan à deux dimensions.	Spécifique à PostgreSQL.
	`lseg`	Segment de ligne fini dans un plan à deux dimensions.	Spécifique à PostgreSQL.
	`circle`	Cercle, défini par son centre et son rayon.	Spécifique à PostgreSQL.
	`path`	Chemins géométriques ouverts et fermés dans un plan à deux dimensions.	Spécifique à PostgreSQL.
	`point`	Point géometrique dans un plan à deux dimensions.	Spécifique à PostgreSQL.
	`polygon`	Chemin géométrique fermé dans un plan à deux dimensions.	Spécifique à PostgreSQL.
Types pour le réseau	`cidr`	Adresse/masque IP.	Spécifique à PostgreSQL.
	`inet`	Adresse IP, avec un masque facultatif.	Spécifique à PostgreSQL.
	`macaddr`	Adresse MAC (adresse d'une carte Ethernet, par exemple).	Spécifique à PostgreSQL.
Types pour le système	`oid`	Identificateur d'objet (ligne).	Spécifique à PostgreSQL.
	`xid`	Identificateur de transaction.	Spécifique à PostgreSQL.

```
SELECT partenaires.id,partenaires.nom from partenaires, clients
WHERE (clients.position<->partenaires.position)<0.2
```

Automatisation des procédures courantes

Les séquences : liste d'incrémentation

```
CREATE SEQUENCE commande_id MINVALUE 0;
SELECT nextval('commande_id'); \\renvoie la dernière valeur de la séquence et incrémente
CREATE TABLE commandes (id integer DEFAULT nextval('commande_id') PRIMARY
KEY, id_client integer, id_produit integer)
```

Les Triggers :

```
CREATE TRIGGER {BERORE|AFTER} {évènement }
ON nom_table
FOR EACH {ROW|STATEMENT}
EXECUTE PROCEDURE nom_fonction (paramètres)
```

Les fonctions SQL :

```
->CREATE FUNCTION isbn_titre(text) RETURNS text
AS 'SELECT titre FROM livres WHERE isbn=$1'
LANGUAGE 'SQL';
->SELECT isbn_titre('099993333');
```

Les fonctions C

```
$gcc -shared est-nul.c -o est_nul.so
->CREATE FUNCTION est_nul(int4) RETURNS Boolean
AS '/home/master/est_nul.so' LANGUAGE 'C';
```

Les fonctions PL/pgSQL (sous ORACLE PL/SQL) :

-> CREATE FUNCTION identificateur (paramètres) RETURNS type AS ‘
DECLARE
Déclaration;
[…]
BEGIN
Instruction;
[…]
END;
‘ LANGUAGE ‘plpgsql’

```
CREATE FUNCTION get_id_client (text,text) RETURNS integer AS '
  DECLARE

    -- Déclaration d'alias pour les paramètres.
    nom_client ALIAS FOR $1;
    prenom_client ALIAS FOR $2;

    -- Déclaration d'une variable pour contenir l'ID du client.
    id_client integer;

  BEGIN

    -- Obtention de l'ID du client dont le prénom et le nom correspondent
    -- aux paramètres de la fonction.
    SELECT INTO id_client id FROM clients
      WHERE nom = nom_client AND prenom = prenom_client;

    -- Renvoie l'ID du client.
    RETURN id_client;
  END;
' LANGUAGE 'plpgsql';
```

```
booktown=# SELECT get_id_client('Jackson','Annie');
 get_id_client
---------------
 107
(1 row)
```

Transactions et curseurs

N'utilise pas la technique des verrous -> effet néfaste sur les performances (consultation bloquée par exemple tant que la transaction n'a pas été validée (COMMIT)

Utilise la technique de MVCC (Multi-Version Concurrency Control) : tant que les modifications n'ont pas été validées, chaque connexion à postgreSQL maintient un instantané temporaire de la base de données pour les objets modifiés dans un bloc transactionnel (BEGIN … COMMIT)

Utilisateur 1 :

->BEGIN;

->UPDATE sujets SET emplacement = NULL WHERE id =132;

Utilisateur 2 :

-> SELECT emplacement FROM sujets WHERE id=12;

Paris

Utilisateur 1 :

-> SELECT emplacement FROM sujets WHERE id=12;

NULL

-> COMMIT;

Utilisateur 2 :

-> SELECT emplacement FROM sujets WHERE id=12;

NULL

Un curseur est un pointeur en lecture seule vers l'ensemble résultat d'une instruction SELECT complètement exécutée : utilisé pour les applications qui maintiennent une connexion persistante vers un serveur PostgreSQL.

En exécutant un curseur et en gérant une référence vers l'ensemble résultat qu'il produit, une application peut gérer plus efficacement les lignes qu'elle souhaite récupérer à différents instants sans devoir exécuter à nouveau la requête avec les clauses LIMIT et OFFSET différentes.

Les curseurs sont souvent encapsulés dans une API bien que l'on puisse également les manipuler directement via les commandes SQL standard. Noter qu'ils sont toujours encapsulés à l'intérieur d'une transaction (normal).

```
-> BEGIN;
-> DECLARE tous_livres CURSOR FOR SELECT * FROM livres;
-> FETCH 4 FROM tous_livres;
-> FETCH NEXT FROM tous_livres;
-> FETCH PRIOR FROM tous_livres;
-> MOVE FORWARD 10 IN tous_livres;
-> CLOSE tous_livres;
-> COMMIT;
```

En C :

Pré-processeur SQL :

$ecpg -o nom.c nom.sql -I /usr/include/psql/

Compilation :

$cc nom.c -o nom -I /usr/include/pgsql -lecpg -lpq

Exécution du programme :

$./nom |more

En Java :

La classe java.sql.ResultSet et ses méthodes

```
/* ================================================================== */
/* Exemple de consultation d'une BD à partir d'un programme C         */
/* Preprocesseur SQL: ecpg -o nom.c nom.sql -I /usr/include/pgsql/    */
/* Compilation: cc nom.c -o nom -I/usr/include/pgsql -lecpg -lpq      */
/* Execution: ./nom                                                   */
/* ================================================================== */

#include <stdio.h>
/* ===== déclaration de variables hôtes    ===== */
/* ===== les types de ces variables sont   ===== */
/* ===== fonction des types SQL dans la BD ===== */

        exec sql begin declare section;
             int etudiant_id;
             [redacted]
        exec sql end declare section;

/* ===== include propre à SQL ===== */
        exec sql include sqlca;

int main(){
        printf("Début du programme\n");

        /* ===== Connexion à la BD ===== */
        exec sql connect to maitrisedb@opale user "user_name" using "password";

        /* ===== déclaration d'un curseur ===== */
           exec sql declare C cursor
              for select
                  from et
                  where
                  [redacted]

        /* ===== ouverture du curseur ===== */
           exec sql open C; /* => Provoque l'envoi de la requête au SGBD = */

        /* ===== positionnement du curseur sur le 1er tuple du resultat = */
          exec sql fetch from C into :etudiant_id, :nom, :prenom, :groupe;

        for (;sqlca.sqlcode==0;){ /* => le fecth value sqlca.sqlcode = */

                printf("%
                [redacted]

              /* ===== positionnement du curseur sur tuple suivant ===== */
                 exec sql fetch from C into
                 [redacted]

         } /* for (;sqlca.sqlcode==0;) */

         /* ===== fermeture du curseur ===== */
         exec sql close C;

         exec sql disconnect;

         exit(0);

} /* main */
```