Cours de Systèmes de Gestion De Bases de Données

Yann Morère

Janvier 2002

Plan

1. Concepts Généraux
2. Modélisation Conceptuelle : Généralités
3. Modélisation Conceptuelle : Le Modèle Entité-Association
4. Le Modèle Relationnel
5. Le Langage SQL

Introduction

Les bases de données = grande place dans l'informatique

- gestion,
- Internet.

30 dernières dernières années, développement de concepts, d'algorithmes, de méthodes ⇨ gérer des données sur mémoires secondaires (disques, bandes etc...).

Essentiel de la discipline «Base de données». Basée sur une théorie fondamentale concernant la modélisation et le traitement des données.

Chapitre 1

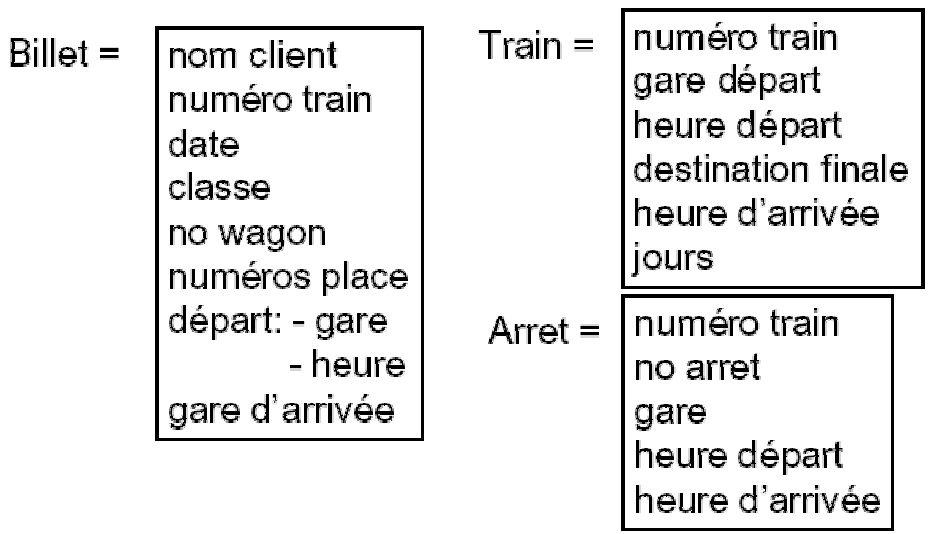
Concepts Généraux

Exemples classiques d'applications BD

- Gestion des personnels, étudiants, cours, inscriptions, ...
- Système de réservation de places d'avion de Air France
- Gestion des comptes clients de Banques
- Gestion des commandes à Amazon.com
- Gestion des jeux olympiques
- ...

Exemple SNCF (fictif)

Gestion des réservations de billets de trains



Besoins de description

1 - Décrire le données de l'application (trains, trajets et réservations) sans faire référence à une solution informatique particulière

⇨modélisation conceptuelle

•2 - Élaborer une description équivalente pour le stockage des données dans le SGBD choisi

⇨ modélisation logique

⇨ langage de description de données (LDD)

Besoins de manipulation

• 3a - Créer la base de données initiale avec les données représentant le réseau SNCF

⇨ langage permettant l'insertion de données

• 3b - Créer au fur et à mesure les données sur les réservations. Modifier si besoin et éventuellement supprimer toute donnée déjà rentrée

⇨ langage de manipulation de données (LMD) (insertion, modification, suppression)

Besoins d'interrogation

- 4 - Répondre à toute demande d'information portant sur les données contenues dans la base. Par exemple:
- a) Durand Julien a-t-il une réservation pour aujourd'hui ?
  - Si oui, donner les informations connues sur cette réservation.
- b) Quels sont les horaires des trains de Metz à Paris entre 9h et 10h le dimanche ?
- c) Donner les destinations au départ de Metz sans arrêts intermédiaires.

⇨ langage de requête (langage d'interrogation)

Besoins d'exactitude / cohérence

- 5 – Il faut pouvoir exprimer toutes les règles qui contraignent les valeurs pouvant être enregistrées de façon à éviter toute erreur qui peut être détectée. Par exemple:
  - Il ne faut jamais donner la même place dans le même train à 2 clients
  - Les arrêts d'un train sont numérotés de façon continue (il ne peut y avoir pour un train donné un arrêt no 3 s'il n'y a pas un arrêt no 2 et un arrêt no 1)
  - La date de réservation pour un train doit correspondre à un jour de circulation de ce train
  - Le numéro de train dans une réservation / arrêt doit correspondre à un train existant
  - L'heure de départ d'une gare doit être postérieure à l'heure d'arrivée dans cette gare
  - L'heure d'arrivée à un arrêt doit être postérieure à l'heure de départ de l'arrêt précédent

⇨ langage d'expression de contraintes d'intégrité

Besoins de garanties

•6 - Il ne faut pas que les informations (par exemple, les réservations) soient perdues à cause d'un disfonctionnement quelconque : erreur de programmation, panne système, panne de l'ordinateur, coupure de courant, …

⇨ garantie de fiabilité

•7 - Il ne faut pas qu'une action faite pour un utilisateur (par exemple, l'enregistrement d'une réservation) soit perdue du fait d'une autre action faite simultanément pour un autre utilisateur (réservation de la même place).

⇨ garantie de contrôle de concurrence

Besoins de confidentialité

- 8 - Toute information doit pouvoir être protégée contre l'accès par des utilisateurs non autorisés
  - en lecture
  - en écriture
- Interdire par exemple aux clients de modifier les numéros des trains ou les horaires ou leur réservation.

  ⇨ garantie de confidentialité (privacy)

Besoin d'efficacité

- 9-10 Le temps de réponse du système doit être conforme aux besoins:
- en interactif : pas plus de 3 secondes
- en programmation : assez rapide pour assumer la charge de travail attendue (nombre de transactions par jour)

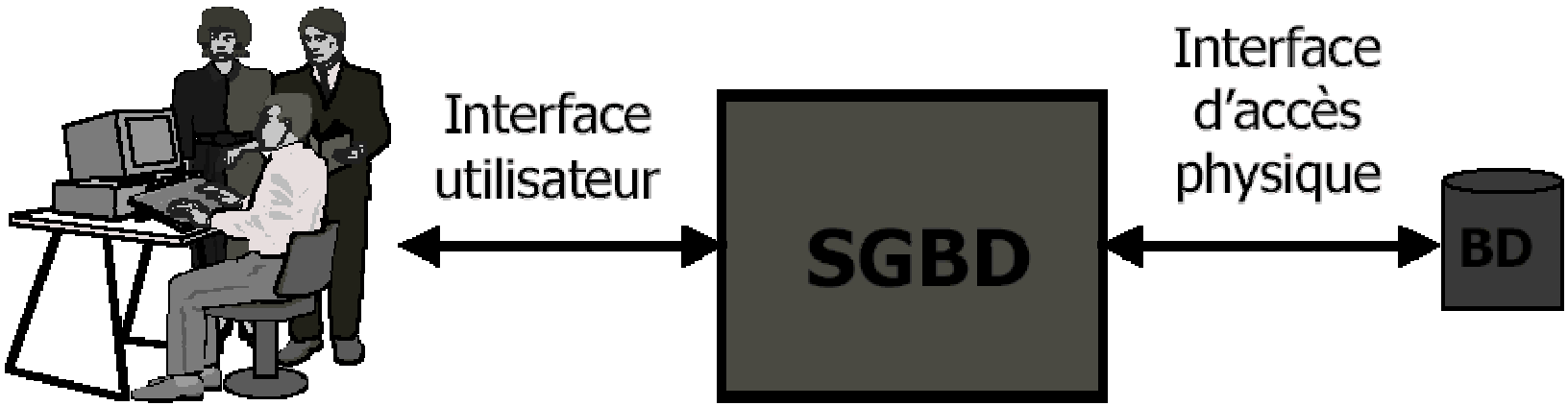
⇨ mécanismes d'optimisation

⇨ éventuellement, répartition / duplication des données sur plusieurs sites

Moyens

- Base de données
  - ensemble cohérent, intégré, partagé de données structurées
  - définie pour les besoins d'une application
- Système de gestion de base de données
  - logiciel permettant de couvrir les besoins :
    - définir une représentation des informations apte à
    - stocker, interroger et manipuler (insérer, supprimer, mettre à jour) de
    - grandes quantités de données (plus que la mémoire vive)
    - dont il faut garantir la longévité et
    - l'accessibilité de manière concurrente (plusieurs utilisateurs simultanés)
    - et sûre.

Architecture d'un SGBD



Analyse/vérification des requêtes

Convivialité de l'interface

Puissance des langages

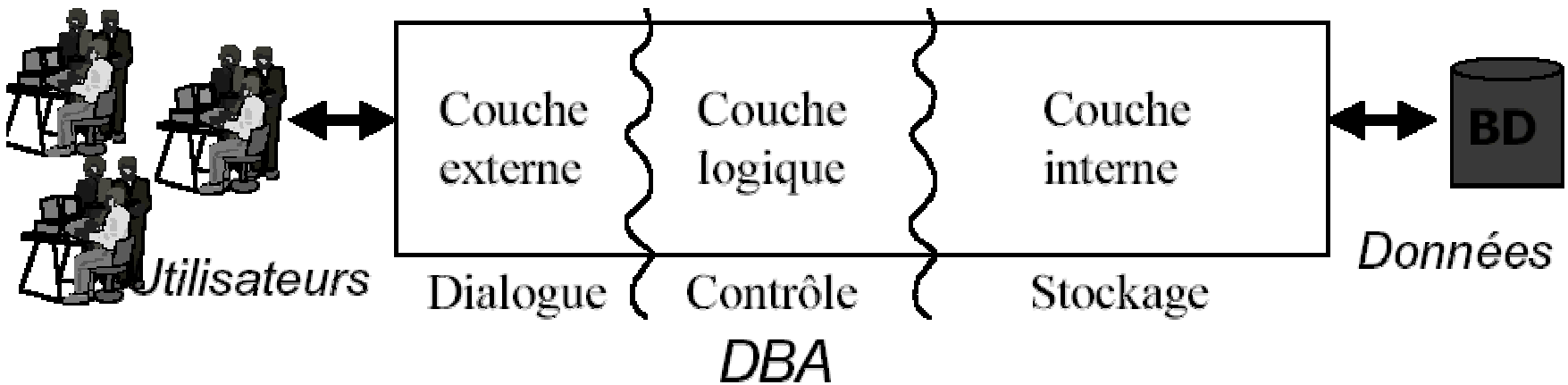
Stockage / accès aux données

Optimisation des performances

SGBD = Boîte noire assurant la gestion de la BD conformément aux requêtes

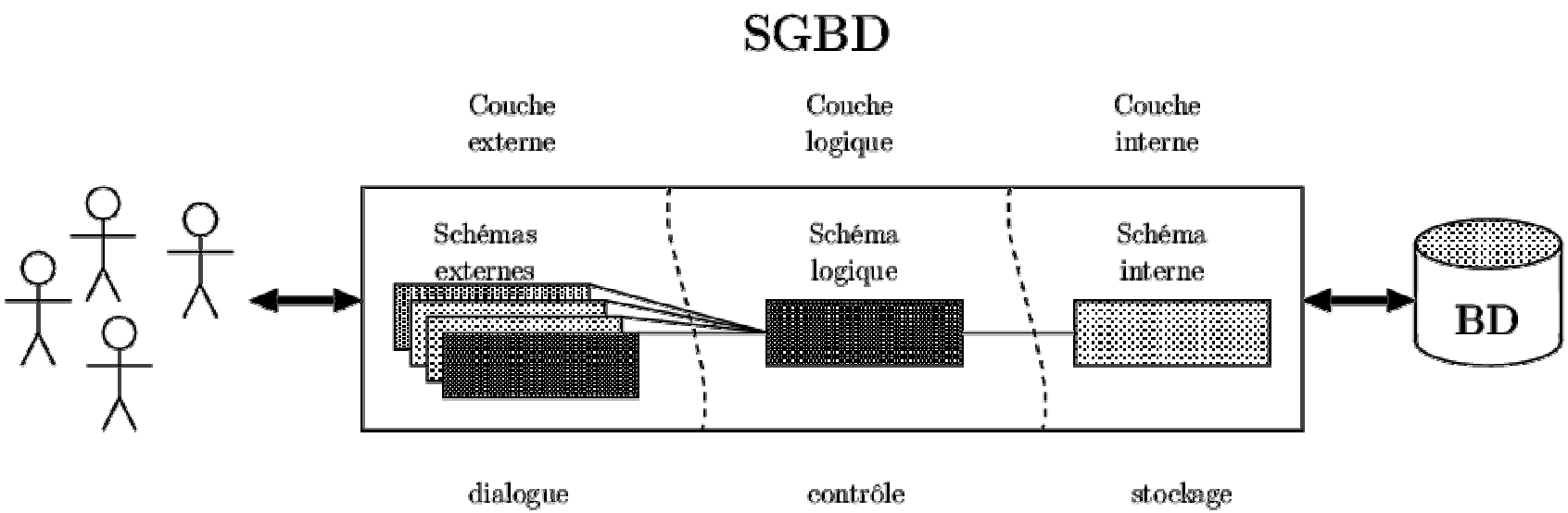
Trois couches

- **Niveau externe :**
  - dialogue avec les utilisateurs
  - vues associées à chaque groupe d'utilisateurs
- **Niveau interne :**
  - stockage des données sur des supports physiques,
  - gestion des structures de mémorisation (fichiers) et d'accès (gestion des index, des clés, ...)
- **Niveau logique :** contrôle global et structure globale des données



Pour chaque couche

- Modèle de données
  - ensemble des concepts qui permettent de décrire les données d'une base et les règles d'utilisation de ces concepts.
- Schémas d'une BD
  - Descriptions d'une base de données obtenues en employant un modèle de données.



Conception : vers le modèle conceptuel

Description des besoins ⇨ modèle conceptuel

- Support du dialogue concepteurs / utilisateurs
- Indépendant de la solution informatique
- Deux parties couvertes par le modèle:
  - statique (la structure des données)
  - dynamique (règles et opérations)
- Contraintes d'intégrité
  - inhérentes aux données ou traduisant les règles des applications.
    e.g. « il ne doit pas y avoir plus de 20 % d'écart entre les salaires des employés d'un même service et d'une même catégorie »

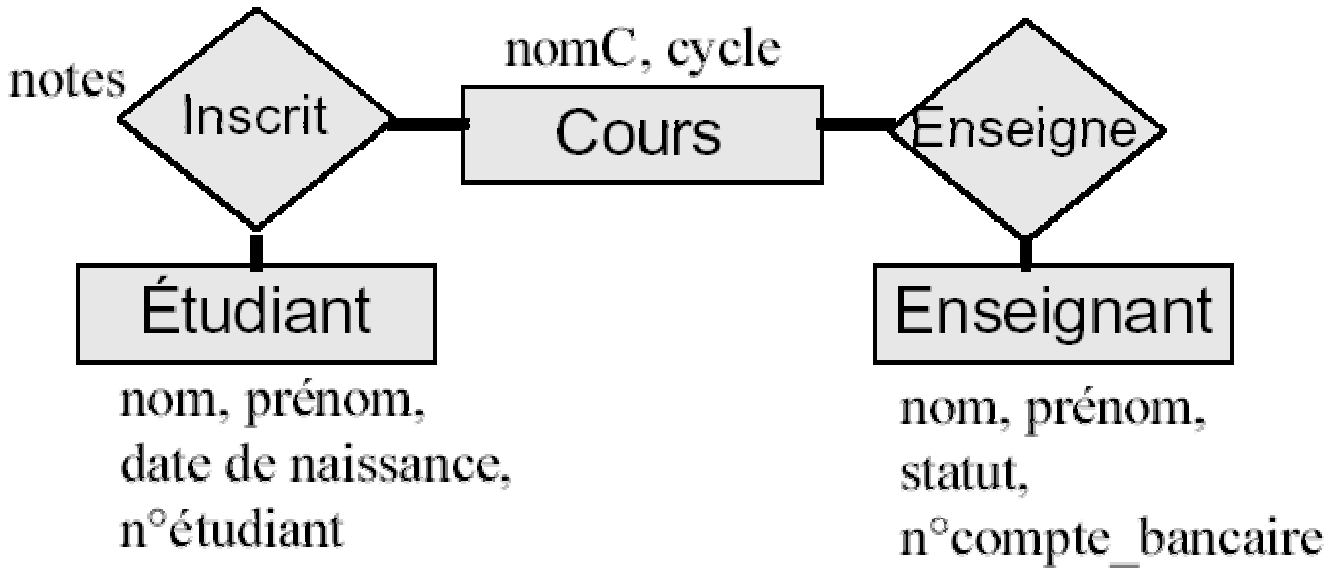
⇨Description de la future base indépendamment des choix techniques

⇨ schéma conceptuel

Exemple

Un institut de formation permanente

Schéma conceptuel (SC) entité-association



Le niveau logique : Implantation

- Traduction du schéma conceptuel en un **schéma logique (SL)** dans les concepts du modèle utilisé par le SGBD choisi
- On appelle **modèle logique** le modèle sur lequel est construit un SGBD.
  - relationnel
  - objet
  - hiérarchique …
- *!!! Attention aux confusions de terminologie entre schéma logique et conceptuel*

Exemple

Un institut de formation permanente

- **Schéma logique (SL) relationnel**
  - Étudiant : nom, prénom, date de naissance, n°étudiant
  - Enseignant : nom, prénom, statut, n°compte_bancaire
  - Cours : nomC, cycle, nom_enseignant
  - Inscription : n°étudiant, nom_cours, note1, note2

Le niveau interne : Implantation des données

- Choix des structures de stockage des données par les administrateurs système
- **Schéma interne (SI)** : description des choix d'enregistrement des données dans les fichiers.
- Fait appel à un nouveau modèle, le **modèle interne**, où les concepts sont ceux de fichier, organisation de fichier, index, chemin d'accès, clé, ...

Schéma interne : exemple

- Étudiant : **fichier** FEtud,
  - contenu : nom, prénom, date de naissance, n°étudiant
    - indexé sur n°étudiant,
    - index secondaire sur nom+prénom
- Enseignant + Cours : **fichier** FEnsCours,
  - contenu : nom, prénom, statut, n°compte_bancaire, liste(nomC, cycle)
    - tel que nom_enseignant dans Cours = nom dans Enseignant
      - indexé sur nom,
      - deux index secondaires, l'un sur nomC, l'autre sur cycle
- Inscription : **fichier** FInscrits,
  - contenu : n°étudiant, nom_cours, note1, note2
    - indexé sur n°étudiant,
    - index secondaire sur nom_cours

Le niveau externe : Utilisation

- Un **schéma externe (SE)** par groupe d'utilisateurs, définissant la vue de la base pour ces utilisateurs
- Avantages de cette approche :
  - simplicité
  - protection (confidentialité)
- Dans les SGBD actuels, le modèle de données employé pour décrire les schémas externes est le même que celui du schéma logique

Schémas externes : le professeur

- Schéma externe du professeur de base de données :
- Étudiant_BD : nom, prénom, note1, note2, note_finale
  - tel que

Étudiant_BD résulte de la combinaison de Étudiant et Inscription du SL,

tels qu'il existe une Inscription de cet étudiant pour le cours BD (n°étudiant dans Étudiant = n°étudiant dans Inscription et nom_cours dans Inscription = 'BD'),

et tel que note_finale = (note1 + note2)/2

Schémas externes : le service personnel

- **Schéma externe du service de gestion du personnel enseignant :**

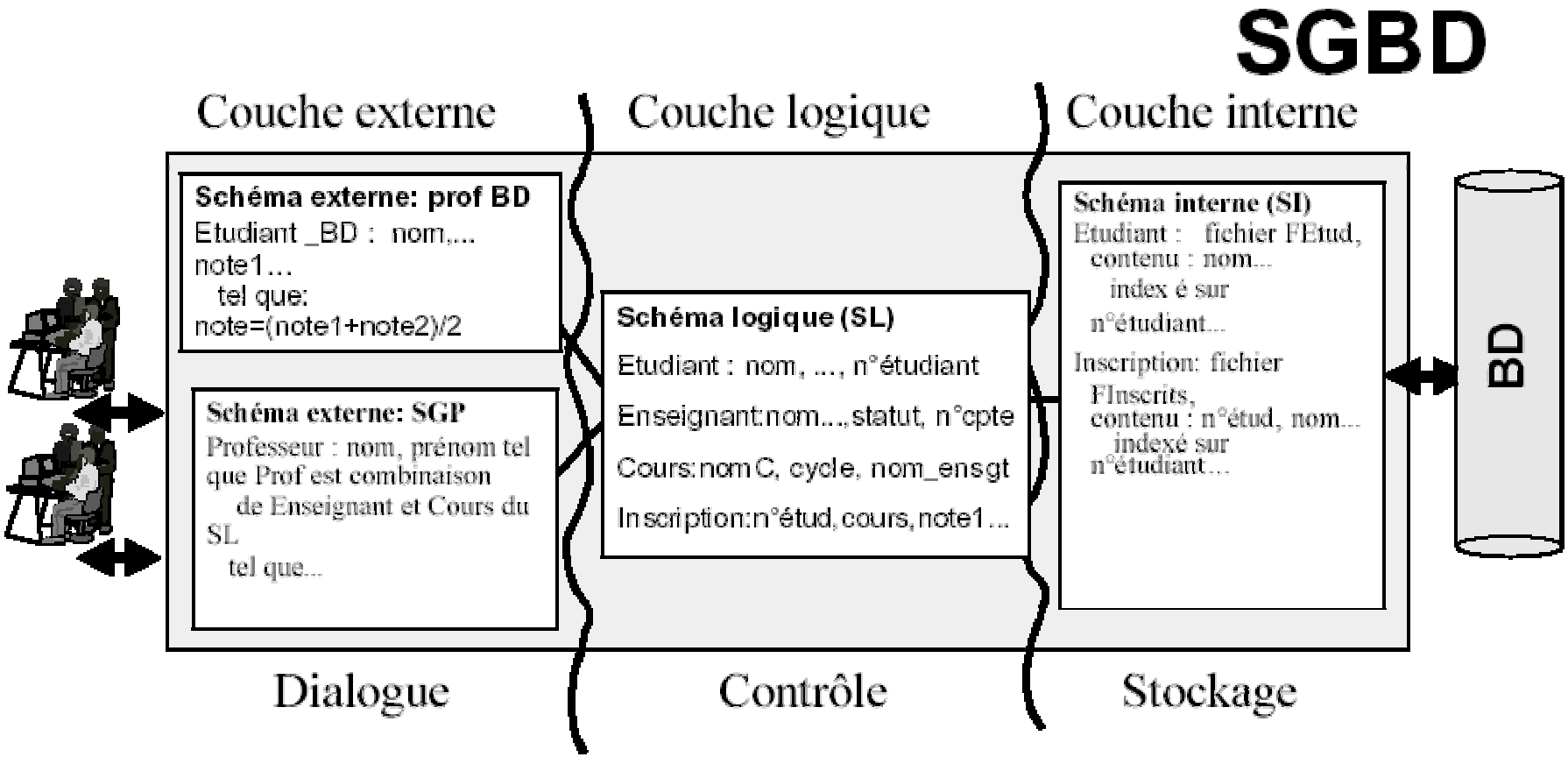
  Professeur : nom, prénom, n°compte_bancaire, nombre_de_cours, liste(nom_cours)

  - tel que Professeur résulte de la combinaison de Enseignant et Cours du SL,

    tels que liste(nom_cours) est la liste de nomC qui se trouvent dans Cours tel que nom_enseignant dans Cours = nom dans Enseignant, et

  - tel que nombre_de_cours = Cardinalité (liste(nom_cours))

Résumé : 3 types de schémas



La BD vue par les utilisateurs, globalement et par l'informaticien.

Principe de fonctionnement du SGBD (utilisation) :

un exemple avec le « parcours » d'une requête

Exprimée en langage accepté par le système (LMD)

- Analyse syntaxique et sémantique d'une requête
- Traduction au niveau logique
- Contrôles de confidentialité, concurrence...
- Si la requête est acceptée, optimisation et découpage en sous-requêtes élémentaires transférées au niveau interne
- Au niveau interne, traduction des sous-requêtes en requêtes physiques correspondantes,
- Retour des résultats à l'utilisateur.

Cycle de vie d'une base de données

4 phases :

- Conception de la base (⇨ SC)
- Implantation des données

  (⇨ SL, SI, population)
- Utilisation (interrogation, mises à jour)

  < développement des programmes d'application >

  (⇨ SEs)
- Maintenance (correction, évolution)

Chapitre 2

Modélisation Conceptuelle : Généralités

Modélisation des données et schémas

- La modélisation est l'activité d'élaboration d'une représentation structurée de la réalité
- Une BD est une représentation de la partie du monde réel qui intéresse les utilisateurs / les applications.
- La modélisation des données est l'élaboration des structures de données pour les données qui seront enregistrées dans une BD.
- La définition de ces structures est consignée dans le schéma de la base de données.

Modélisation conceptuelle

- **Objectif :** représenter la réalité telle qu'elle est perçue par les utilisateurs
- Le processus de modélisation et la définition de son résultat sous forme d'un schéma conceptuel est appelé conception de la base de données
- La qualité de la conception de la BD est un facteur critique de réussite

Modélisation conceptuelle : avantages

- Attention portée sur les applications
- Indépendante des technologies
  - Portabilité
  - Longévité
- Orientée utilisateur (compréhensibilité, support du dialogue concepteurs / utilisateurs, permet la collaboration et la validation par les utilisateurs)

Autres avantages

- Spécifications formelles, non ambiguës, proches de la vision utilisateur
- Puissance des concepts
- Support d'interfaces visuelles (lisibilité)
  - Diagrammes de définition de données
  - Manipulation de données
- Facilite les échanges d'informations entre SGBD différents

Modèle de données (rappel)

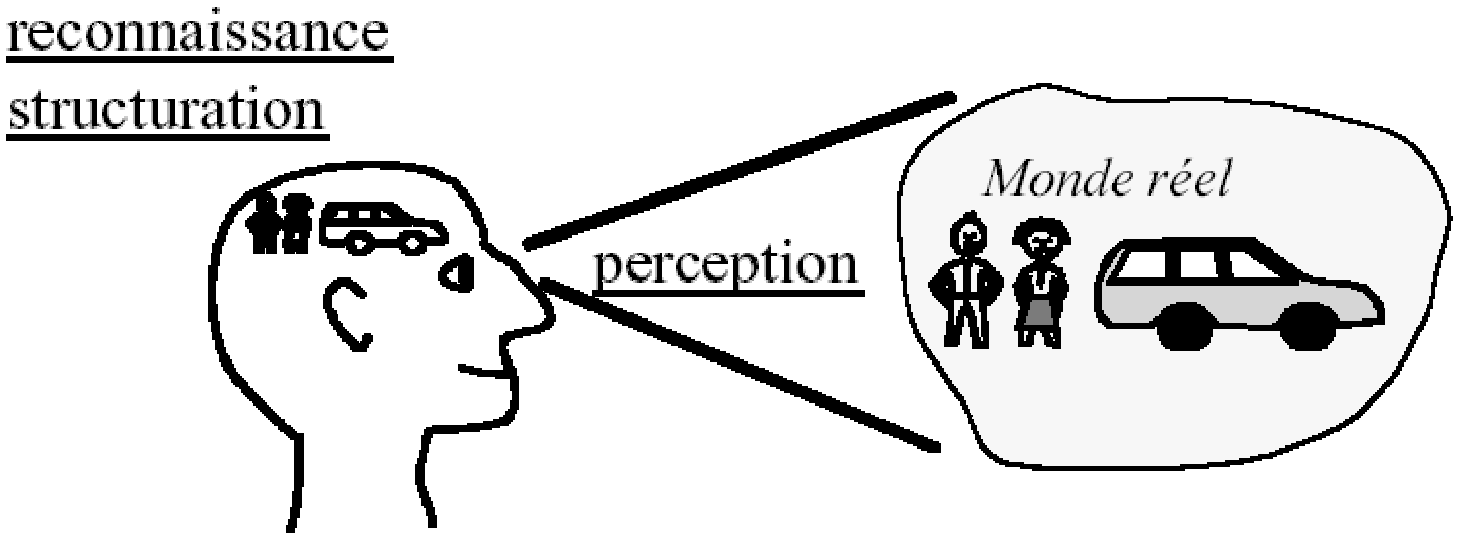
- Ensemble de :
  - concepts permettant la description et la manipulation des données du monde réel
  - règles d'utilisation de ces concepts
- Ces concepts décrivent les aspects:
  - Statiques : structure des données
  - Dynamiques : opérations sur les données

  + contraintes explicites (contraintes d'intégrité)

Modèles conceptuels

- Respectent la trilogie de base
  - objets
  - liens
  - propriétés
- Permettent des représentations multiples

Conception d'une BD : analyse

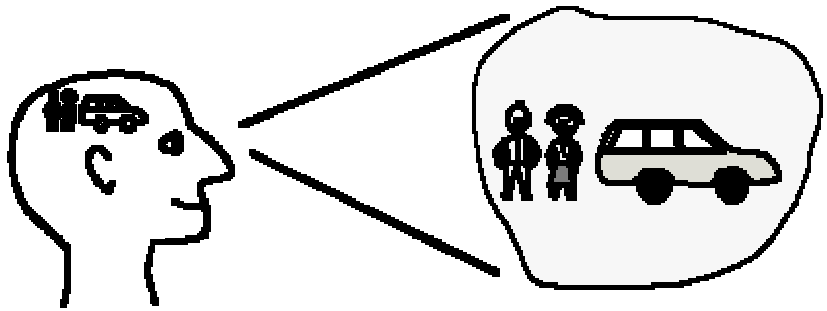


Une BD est une représentation de la partie de la réalité qui nous intéresse.

Conception d'une BD : description

Conception d'une BD : phases

- analyse de la réalité
  - partielle
  - subjective
  - infidèle



- représentation (modèle)
  - contenu
  - structure
  - règles
  - dynamique
- description (langage de définition des données - LDD)

Structure perçue du monde réel

- Jean possède une Honda CRV grise
- Arlette, sa femme, est avec lui

Observations

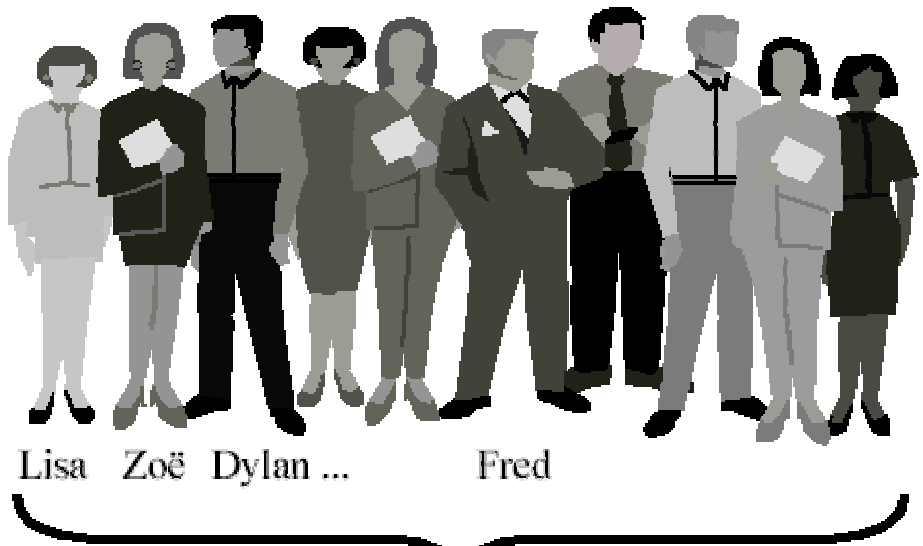
- "Jean" , "Arlette" désignent des objets reconnus comme des Personnes
- "Honda CRV": nom utilisé pour désigner une Voiture
- "Jean" Possède "Honda CRV" : exprime un lien entre une personne et une voiture
- ... est marié avec ... : exprime un lien entre une personne et une autre personne
- "grise": valeur pour la propriété couleur de la voiture
- "Jean": valeur pour la propriété prénom de la personne

Abstraction

De la réalité perçue

à la représentation :

Faire abstraction des particularités permet de passer des objets aux types d'objets



Type d'objet : Personne

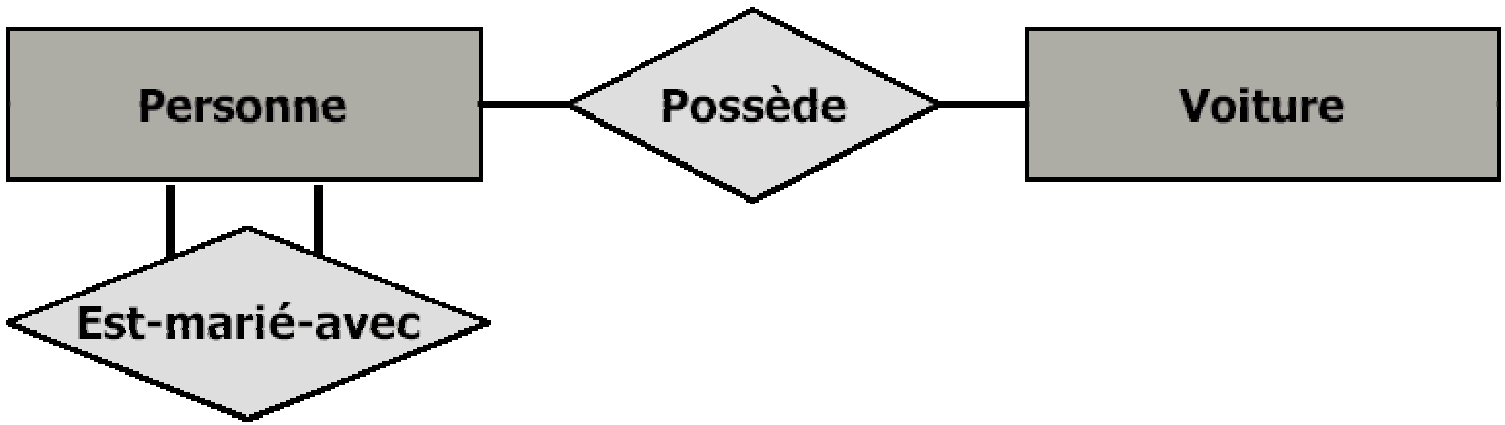
Propriétés : - nom,

- âge, ...

Définition du schéma

Un schéma est une collection de types

La bases de données contiendra les valeurs représentant les instances de ces types



Qualités pour modèles conceptuels

- Complétude

  ⇨ Description de tous phénomènes courants

Fiabilité

  ⇨ formellement défini

- Orientation utilisateur

  ⇨ compréhensible, clair, lisible

- Orthogonalité

  ⇨ indépendance des concepts

- Implémentabilité

  ⇨ traduisible en SGBD existant

- Complètement opérationnel

  ⇨capacités de manipulation des données

Quels modèles conceptuels ?

- Idéalement: tous les concepts utiles
- Pratiquement: un nombre limité
- Base:

  objets + liens + propriétés

  + multi-représentation

  + contraintes d'intégrité

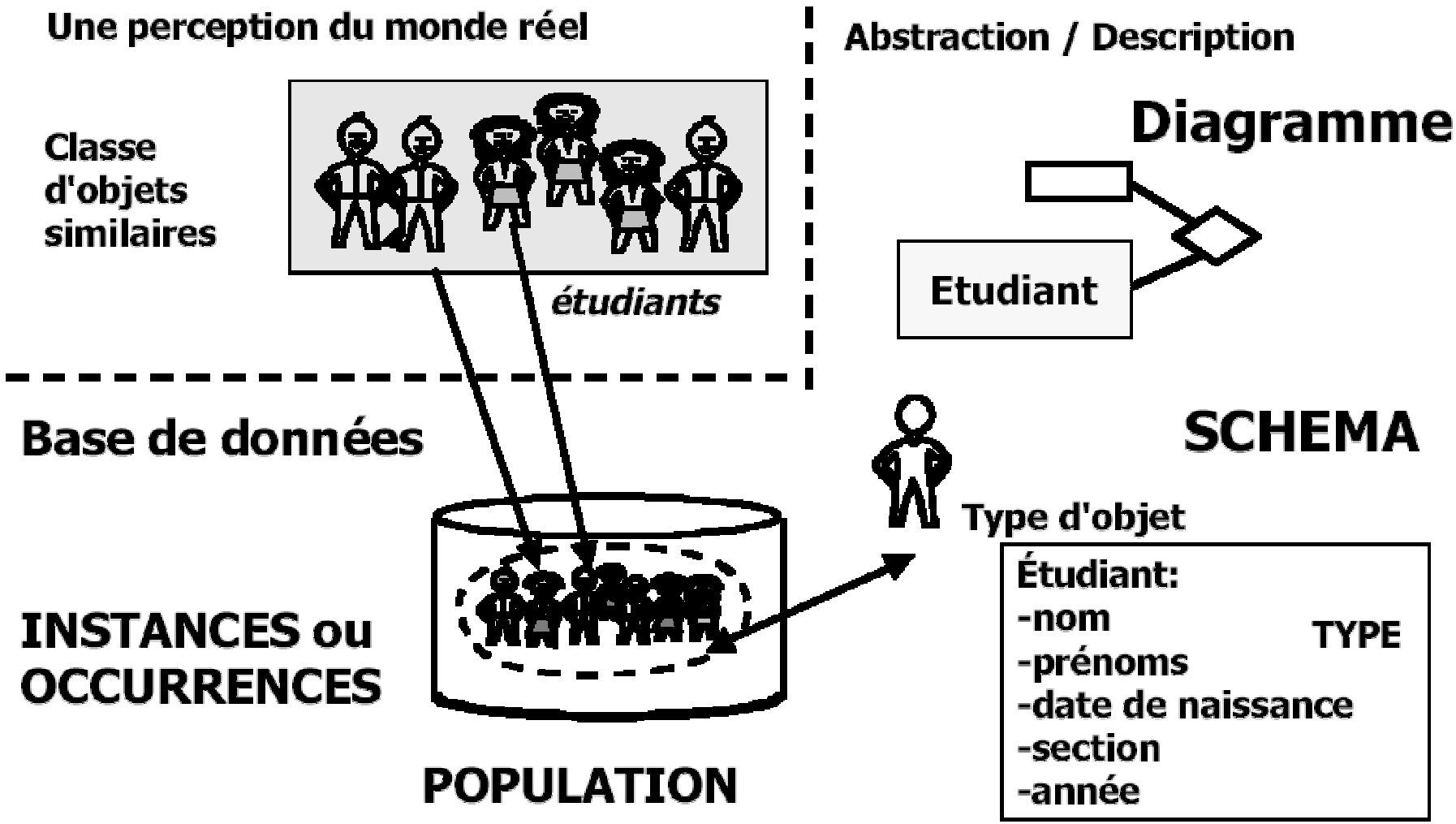
Modèles conceptuels

- Entité-Association (EA)

  (ER: Entity-Relationship)
- UML
- Autres

NB: les modèle relationnels et certains modèles orientés objets sont des modèles logiques (objectif: implémentation)

Quelques termes



Chapitre 3

Modélisation Conceptuelle : Le Modèle Entité-Association

Modèle de type conceptuel

But : permettre la description conceptuelle des structures de données d'une application

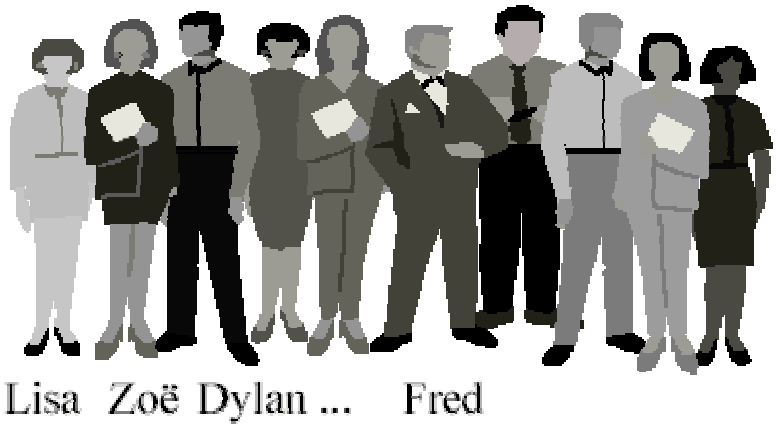
Les concepts de base (correspondent aux concepts d'abstraction de la réalité) :

- objet ⇦⇨ entité
- lien ⇦⇨ association (relationship)
- propriété ⇦⇨ attribut

+ la représentation multiple

Entités et types d'entités

Entité: représentation d'un objet du monde réel ayant une existence propre



Type d'entité (TE):

représentation d'un ensemble d'entités perçues comme similaires et ayant les mêmes caractéristiques

Personne

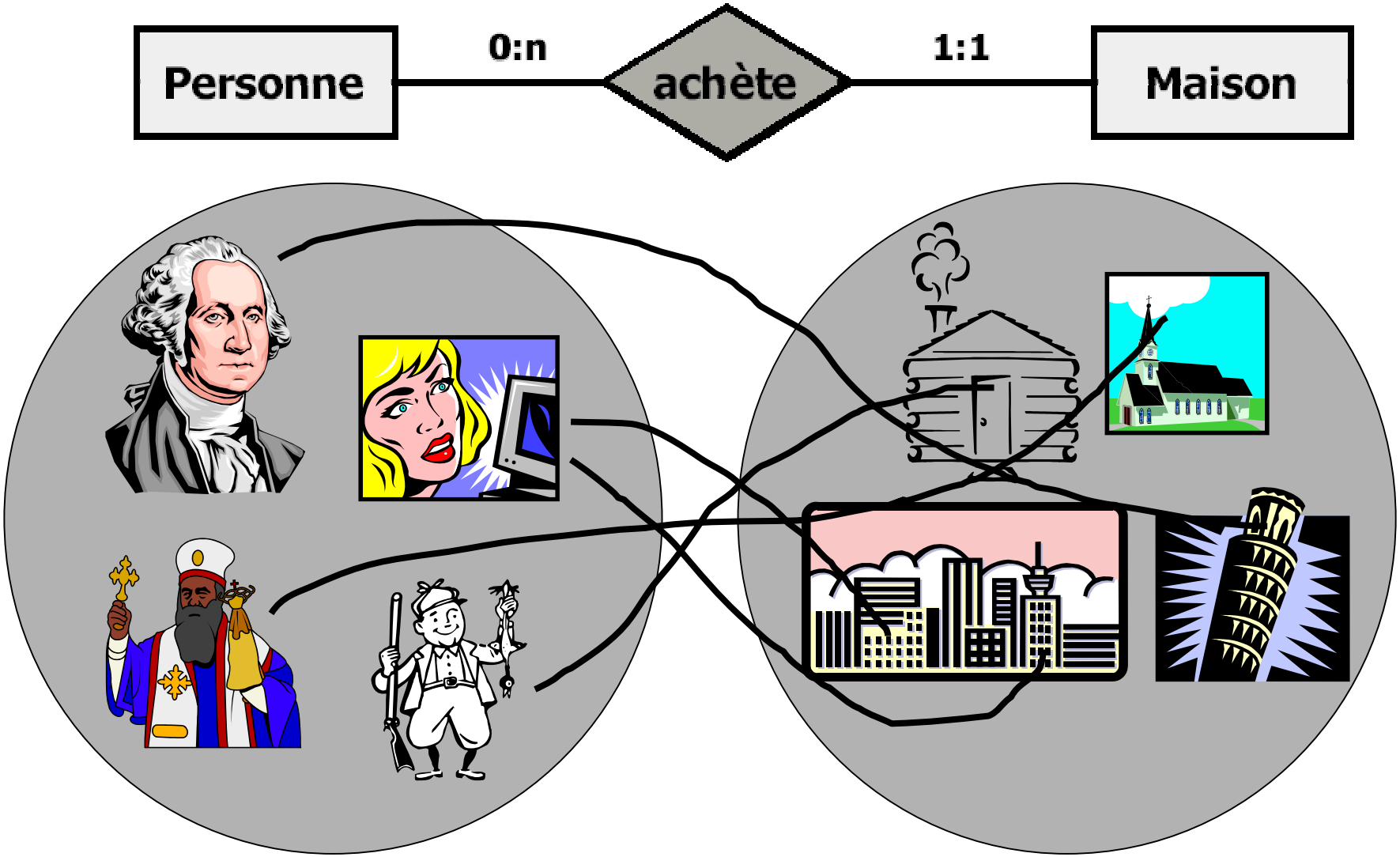
Associations et types d'associations

Association : représentation d'un lien non orienté entre plusieurs entités (qui jouent un rôle déterminé)

Type d'association (TA) : représentation d'un ensemble d'associations ayant la même sémantique et décrites par les mêmes caractéristiques

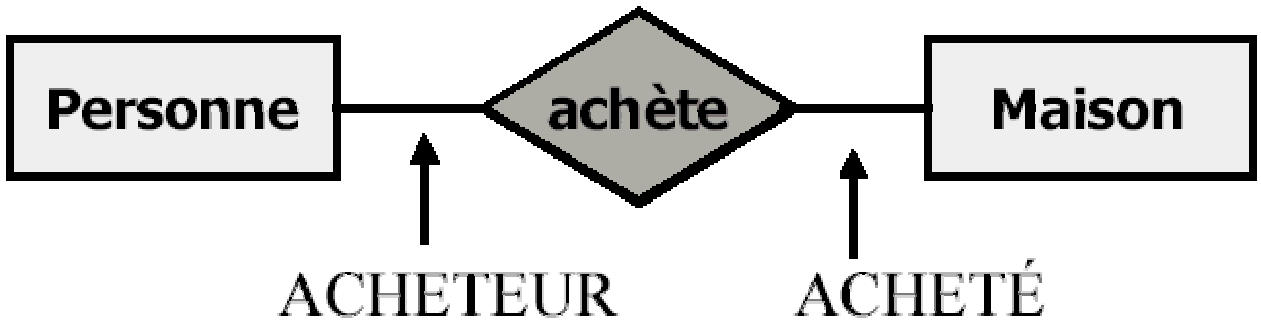
« achète » = < 1 personne, 1 maison >

Population d'un TA



Rôles

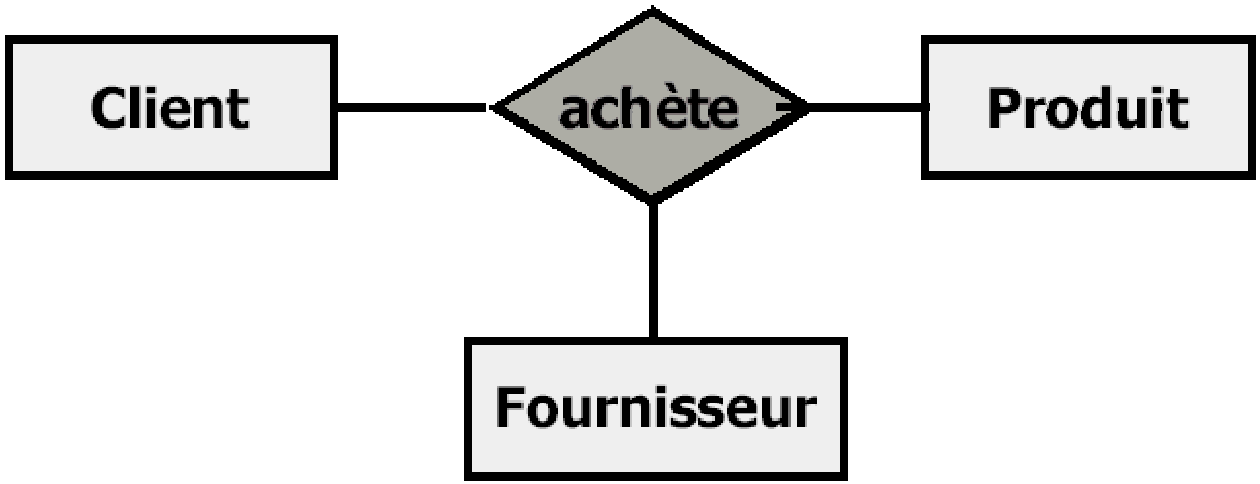
« achète » = < 1 personne, 1 maison >



Association binaire: deux rôles

Associations Ternaires

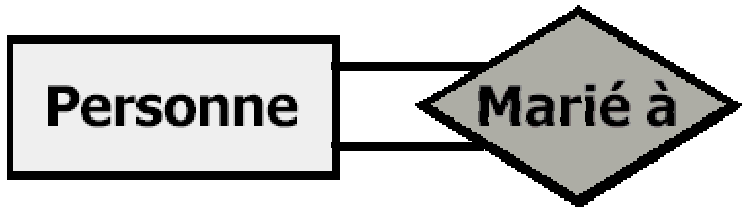
3 rôles



« achète » = < 1 client, 1 produit, 1 fournisseur >

Associations Cycliques

2 rôles (au moins) lient le même type d'entité



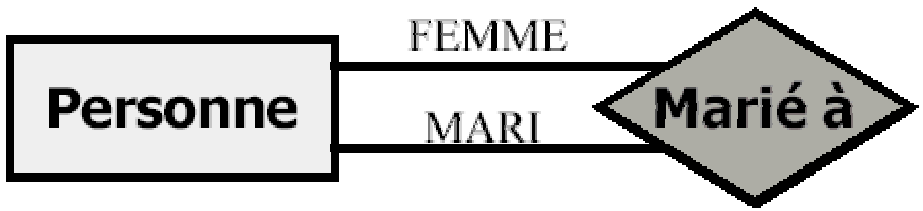
« marié à » = < 1 personne, 1 personne >

Problème : comment savoir dans un couple qui est le mari et qui est la femme ?

< Dupont Dominique, Dupont Dominique > ?

Associations Cycliques: rôles nommés

Solution : spécifier le rôle de chaque entité pour supprimer les ambiguïtés

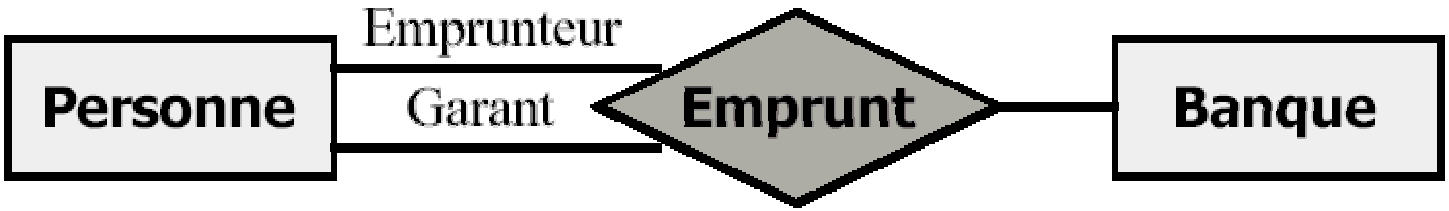


« marié à » = < 1 personne/FEMME, 1 personne/MARI >

< Dupont Dominique / femme, Dupont Dominique / mari >

Associations Cycliques

- Ternaires :



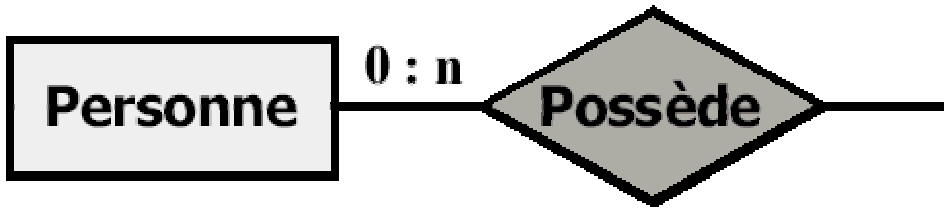
- Symétriques:

Cardinalité des rôles

- Combien de voitures (minimum) une personne peut-elle avoir?
- Combien de voitures (maximum) une personne peut-elle avoir?

Contraintes de cardinalité

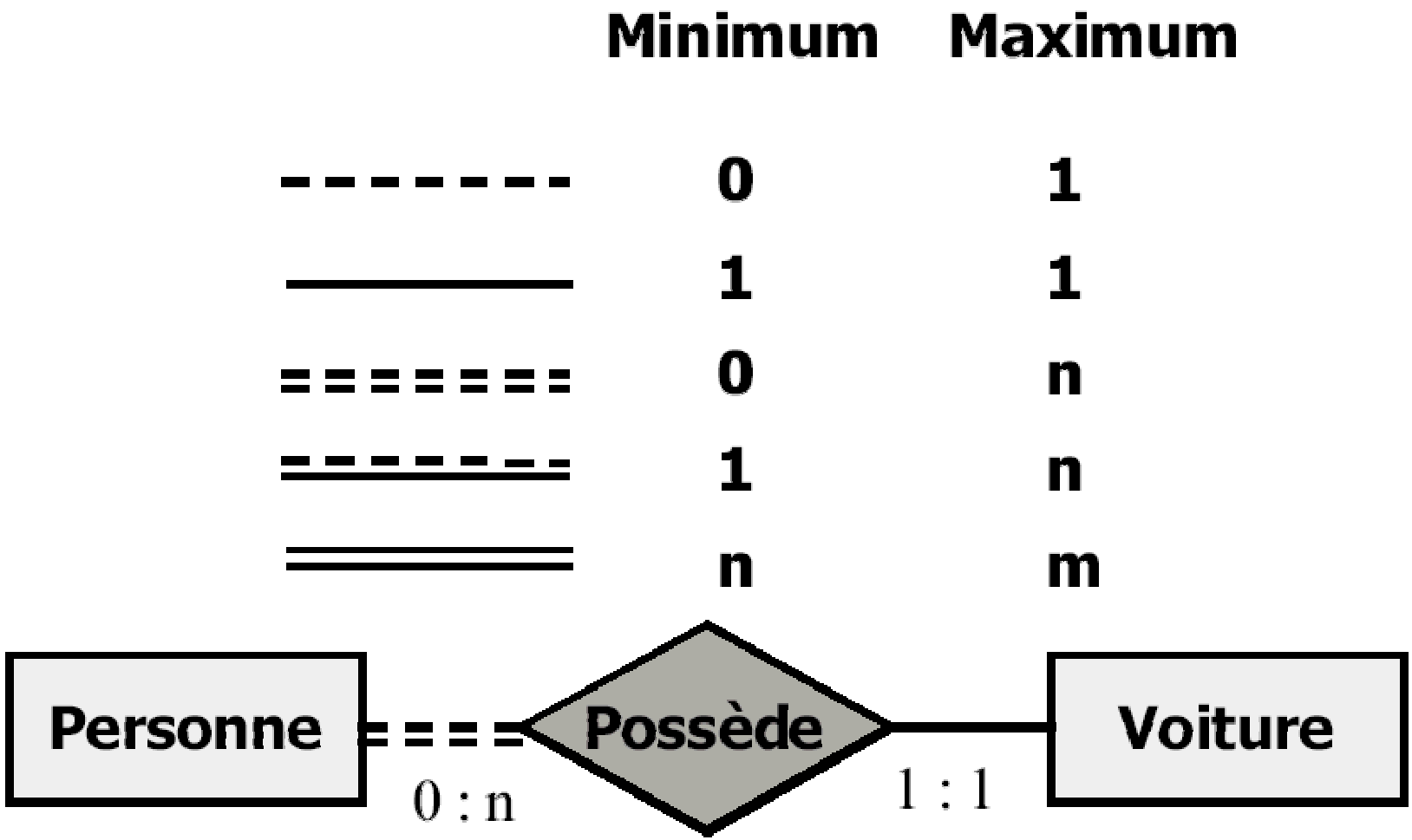
- Une personne peut ne pas avoir de voiture, en avoir 1, 2, … n (pas de contrainte)



- Une voiture a un et un seul propriétaire

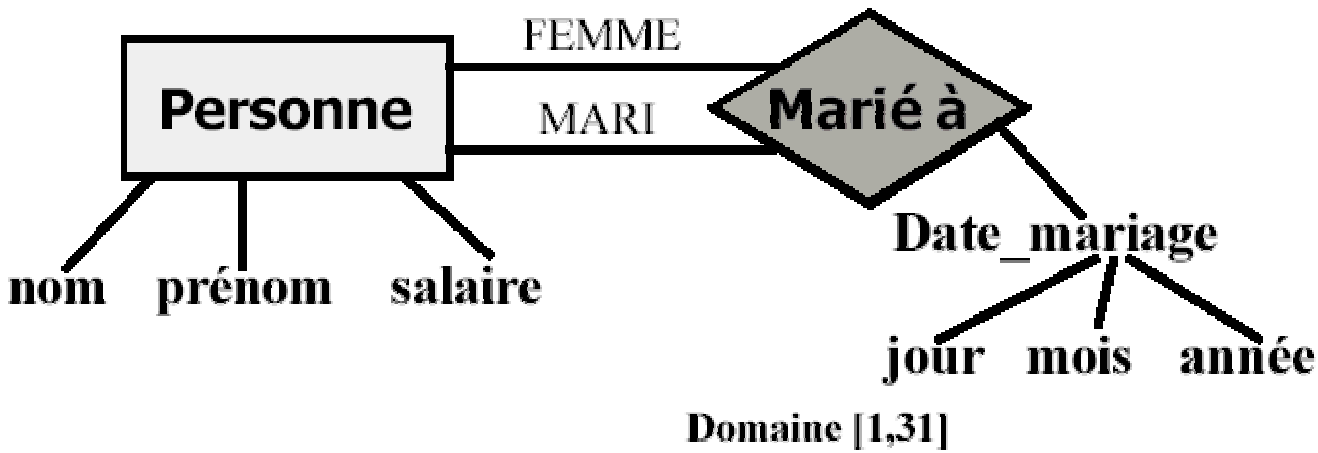
Contraintes sur les cardinalités

Valeurs et notations des cardinalités



Attributs

- Décrivent l'information (les propriétés) à conserver sur :
  - un objet
  - une association
  - un attribut



Attributs simples

- **simple (atomique)**: non décomposable
  - Exemples: jour, prénom
- Feuilles de l'arbre des attributs : seuls les attributs simples portent des valeurs
- Le domaine de valeurs est constitué de valeurs atomiques
  - Ex.: jour - domaine de valeurs: {1, 2, .., 31}
  - Domaines prédéfinis standard, intervalles, énumérés

Attributs complexes

- **complexe**: décomposé en d'autres attributs
  - Exemples: date (jour, mois, année), adresse (rue, ville, code postal)
- Un attribut complexe ne porte pas de valeur propre (pas de domaine directement associé)
- La valeur d'un attribut complexe est la composition des valeurs de ses attributs composants.
- Un composant d'attribut complexe peut être lui-même un attribut complexe.

Attributs mono- ou multivalués

- **monovalué**: une seule valeur par occurrence (cardinalité max=1)

Exemples: date de naissance, numéro sécu

- **multivalué**: plusieurs valeurs par occurrence (cardinalité max>1).

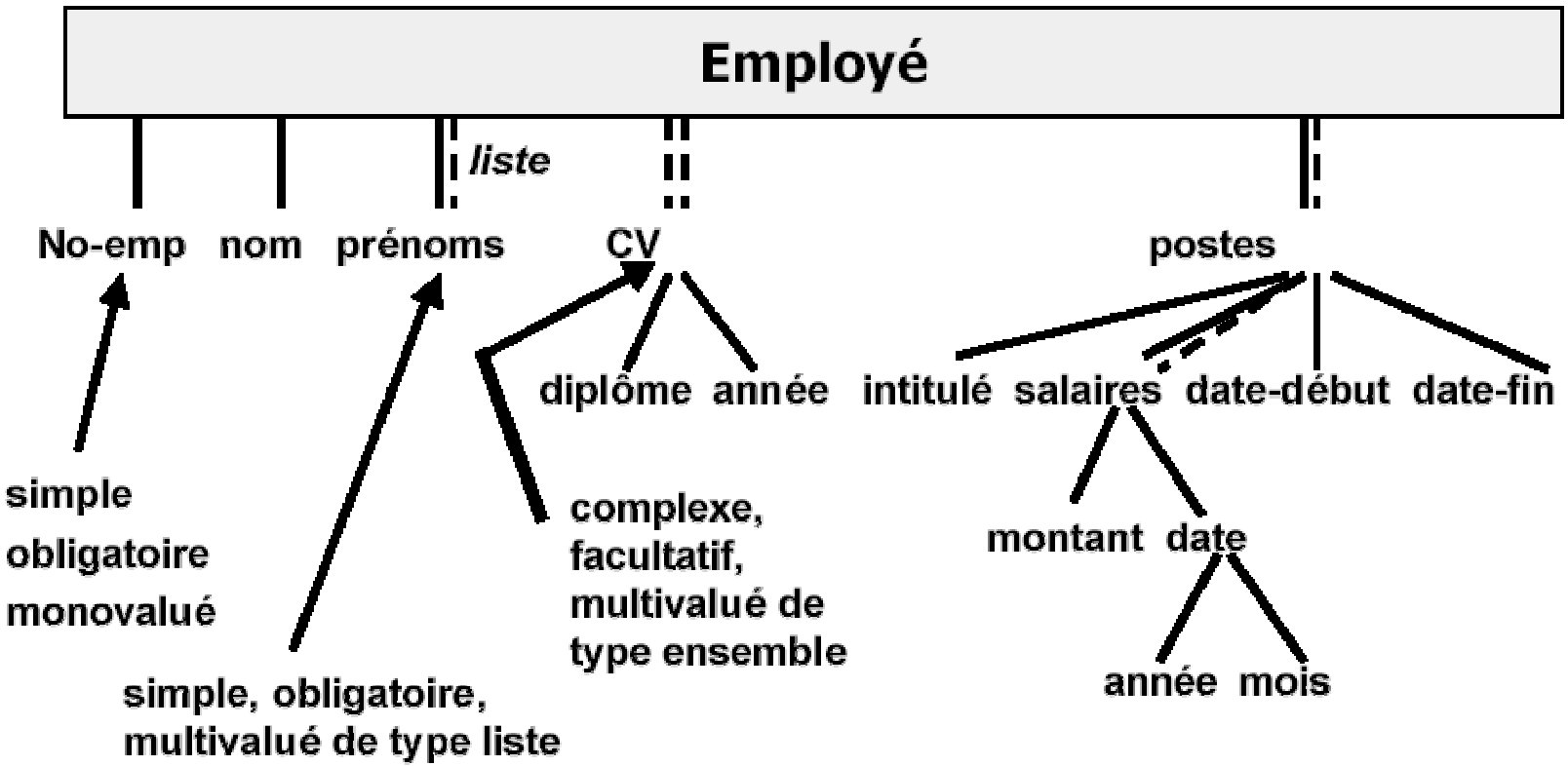
Exemples: prénoms, téléphones

Une valeur d'attribut multivalué est un ensemble (ou liste ou multi-ensemble) de valeurs, prises chacune dans le domaine de valeurs associé à l'attribut.

Attributs obligatoires ou facultatifs

- obligatoire: une valeur au moins par occurrence (cardinalité min>=1).
  - Exemples: nom, prénoms
- facultatif: peut ne pas prendre de valeur (cardinalité min=0).
  - Exemples: salaire, téléphones
- Le caractère obligatoire ou facultatif est déterminé par les besoins de l'application:
  - Si l'on accepte d'enregistrer une personne sans connaître sa date de naissance, alors l'attribut date-de-naissance sera facultatif; sinon, il sera obligatoire

Attributs: exemple



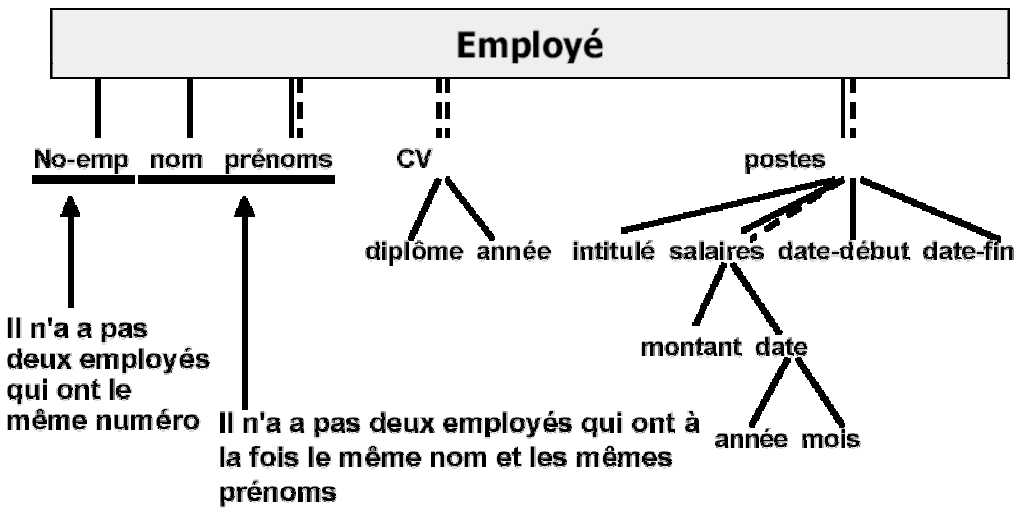
Identifiants de TE et TA

Nécessité de pouvoir désigner une entité (une association) de façon univoque

- Identifiant:
  - Ensemble minimal d'attributs tel qu'il n'existe pas deux instances du TE (TA) où ces attributs aient la même valeur

Identifiants du TE Employé

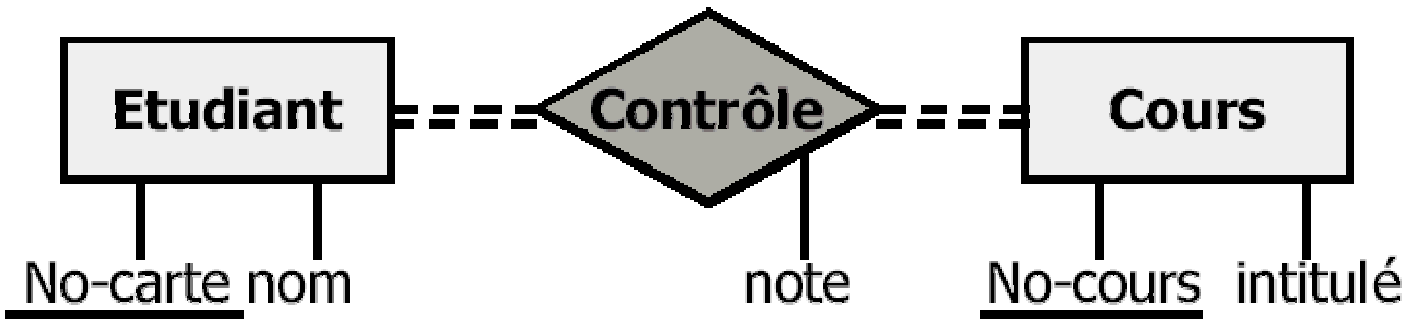
Deux identifiants de Employé: No-emp, nom+prénoms



Identifiant d'un TA: rôles multivalués

- Cas fréquent:

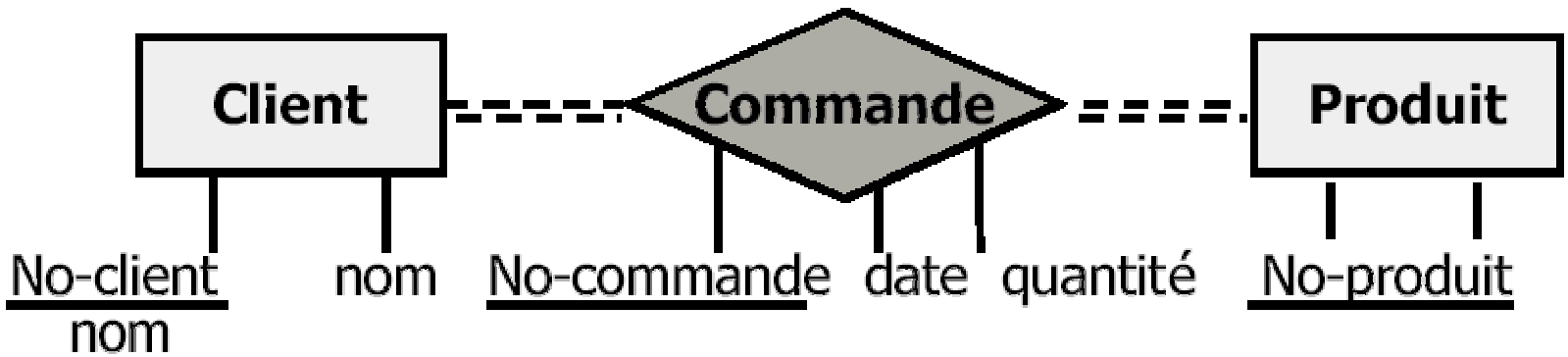
  id.TA = ensemble des identifiants des TE liés



Identifiant de Contrôle : Etudiant.No-carte + Cours.No-cours

Identifiant d'un TA: attribut propre

- id.TA = attribut du TA



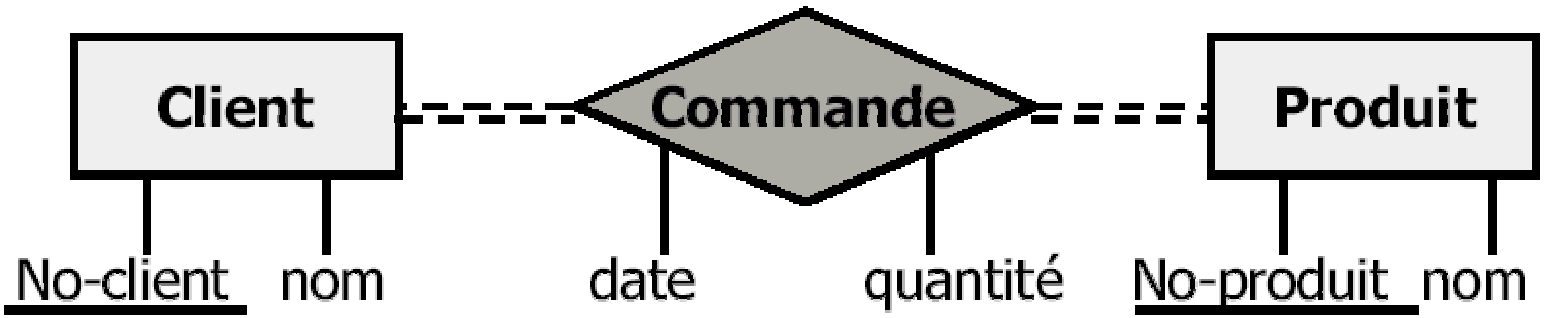
Deux identifiants pour Commande:

- 1) No-commande
- 2) Client.No-client + Produit.No-produit

  (il n'existe qu'une seule commande d'un client donné pour un produit donné)

Identifiant d'un TA: id.TE +attribut propre

- Un client peut commander le même produit plusieurs fois à des dates différentes



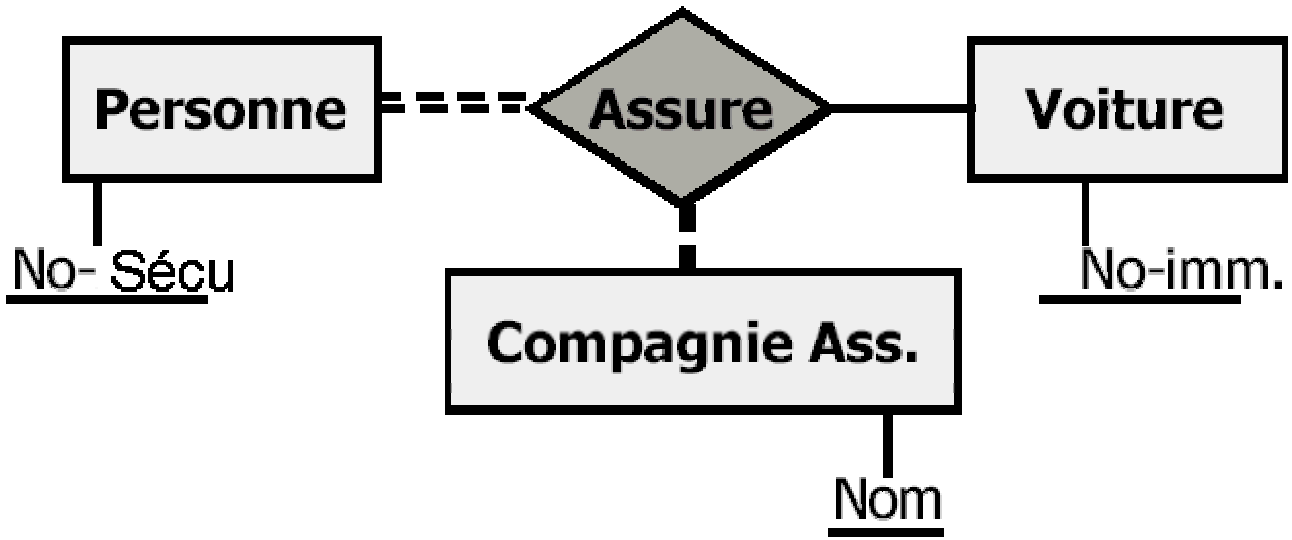
Identifiant de Commande:

Client.No-client + Produit.No-produit + Commande.date

Identifiant d'un TA: rôle monovalué

- Règle:

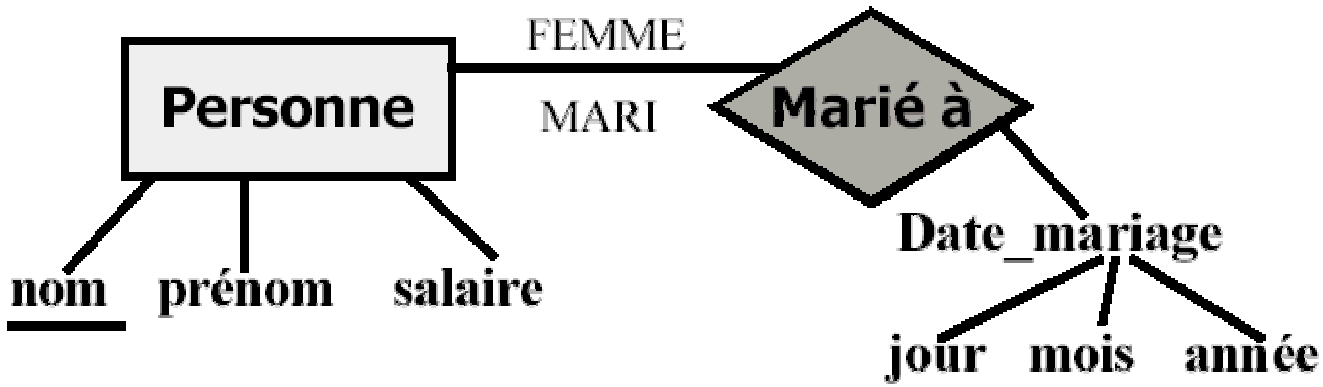
  Tout rôle monovalué induit un identifiant du TA
  (l'identifiant du TE lié est aussi identifiant du TA)



Identifiant de Assure: Voiture.No-imm.

Identifiant d'un TA cyclique

- Comme pour les autres TA



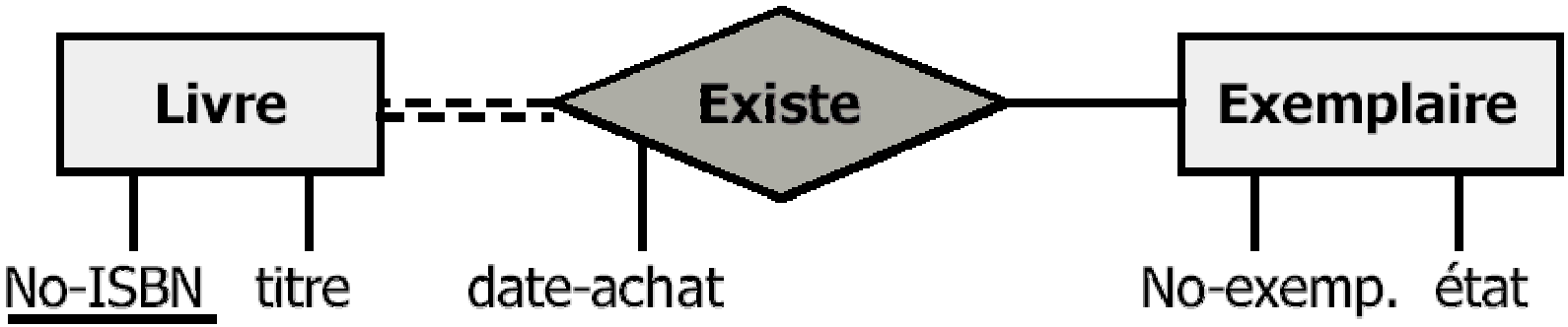
Deux rôles monovalués:

deux identifiants pour Marié à:

- 1) Personne/Femme.nom
- 2) Personne/Mari.nom

Identifiant de TE faible

- Un TE qui ne peut être identifié par ses seuls attributs propres est appelé TE faible

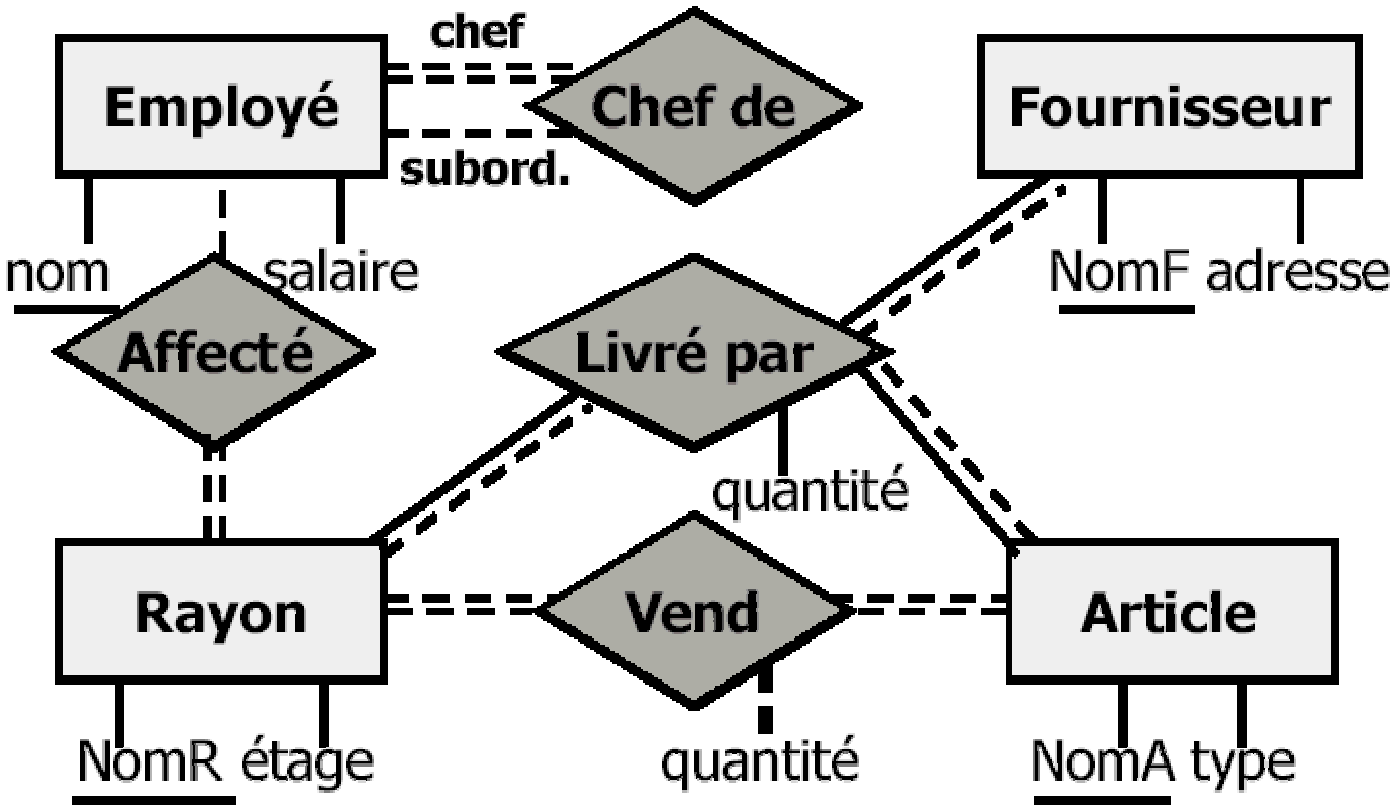


Identifiant de Exemplaire: (Livre.No-ISBN + No-exemp.)

Identifiant de Existe: (Livre.No-ISBN + No-exemp.)

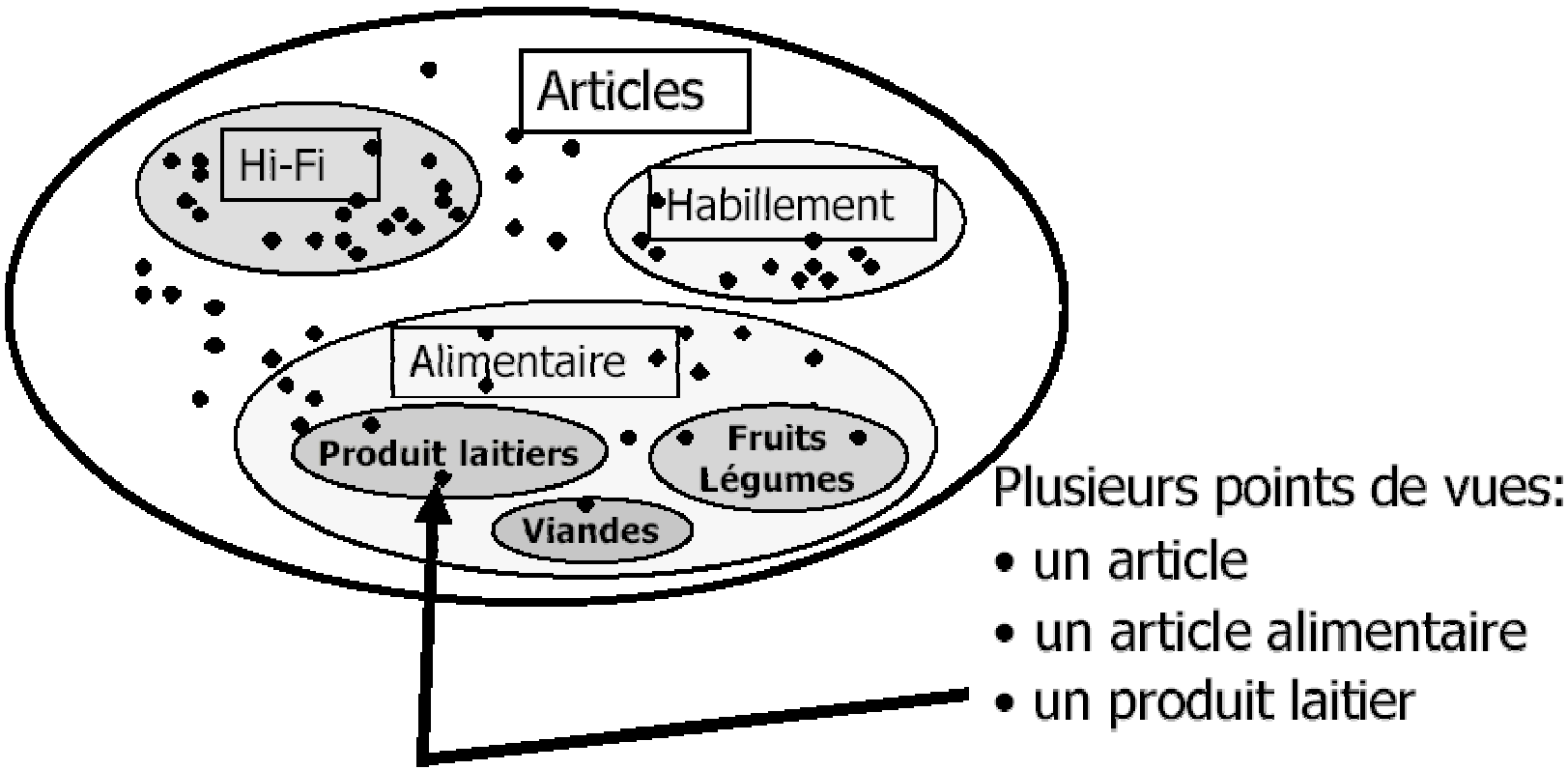
Exemple de schéma EA

Gestion d'un hypermarché

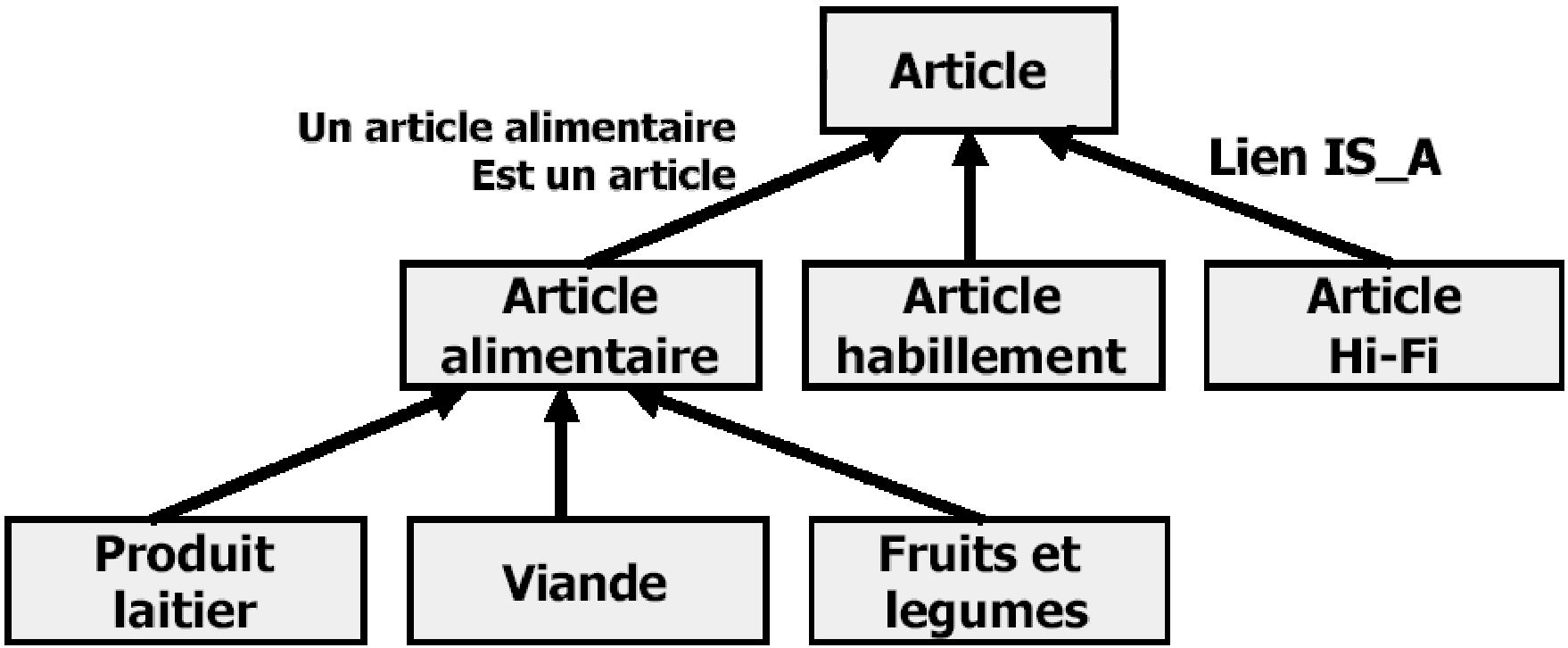


Représentation multiple

- Un objet peut avoir plusieurs représentations

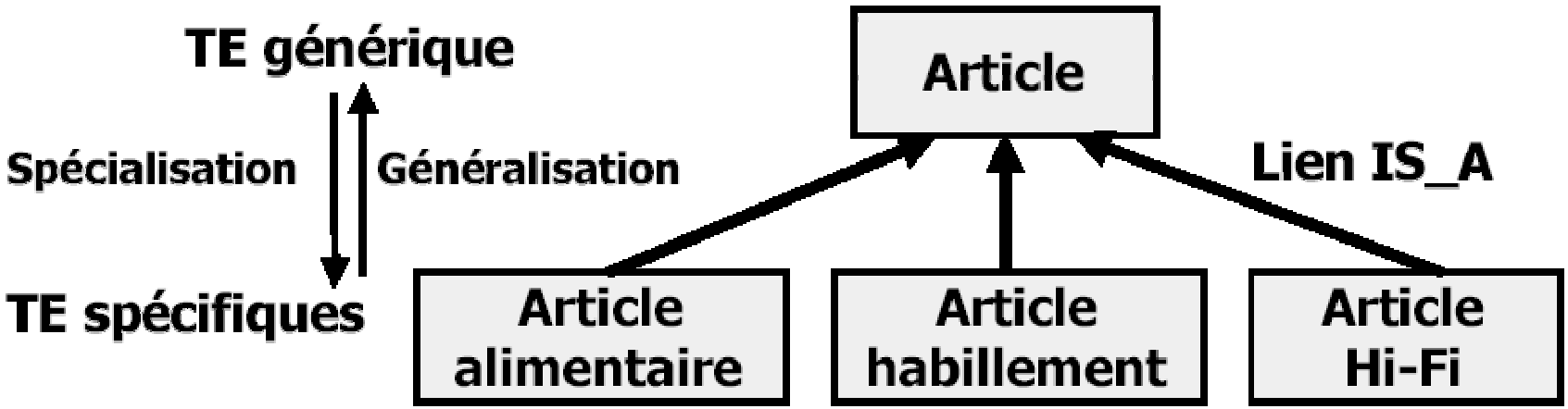


Lien de Généralisation / Spécialisation



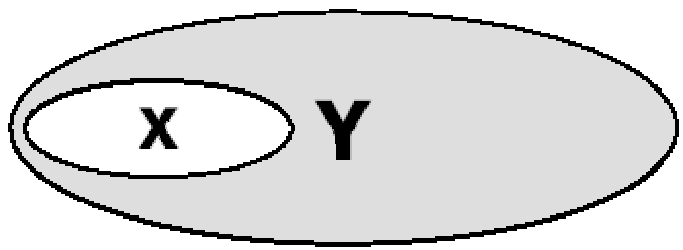
Raffinement de classification

Hiérarchie de Généralisation/Spécialisation

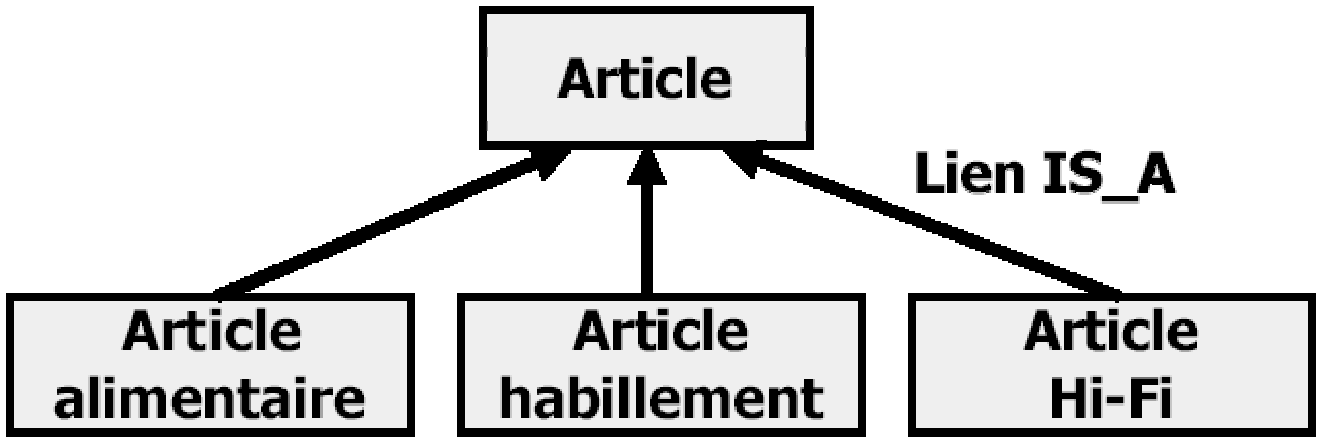


X Est un Y
X sous-type de Y
Y sur-type de X

Inclusion de populations:
tout X est un Y



Contraintes d'intégrité sur Is-a

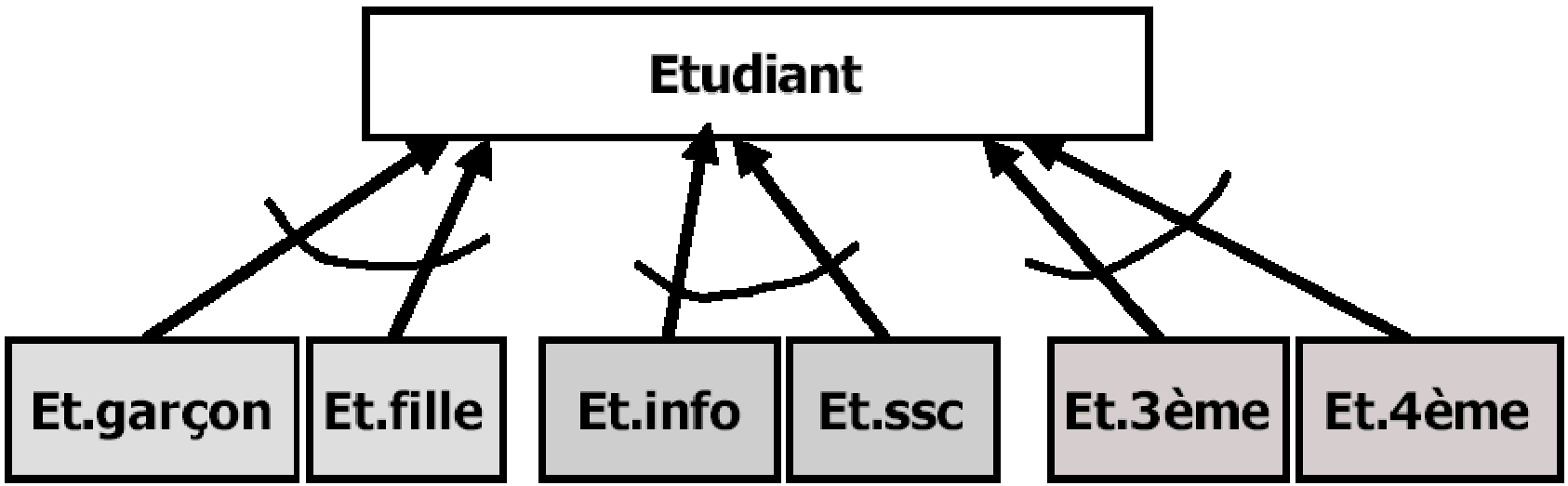


Disjonction: les articles alimentaires et les articles d'habillement n'ont pas d'instances communes

Couverture: tout article appartient à l'un des sous-types (alimentaires, d'habillement ou Hi-Fi)

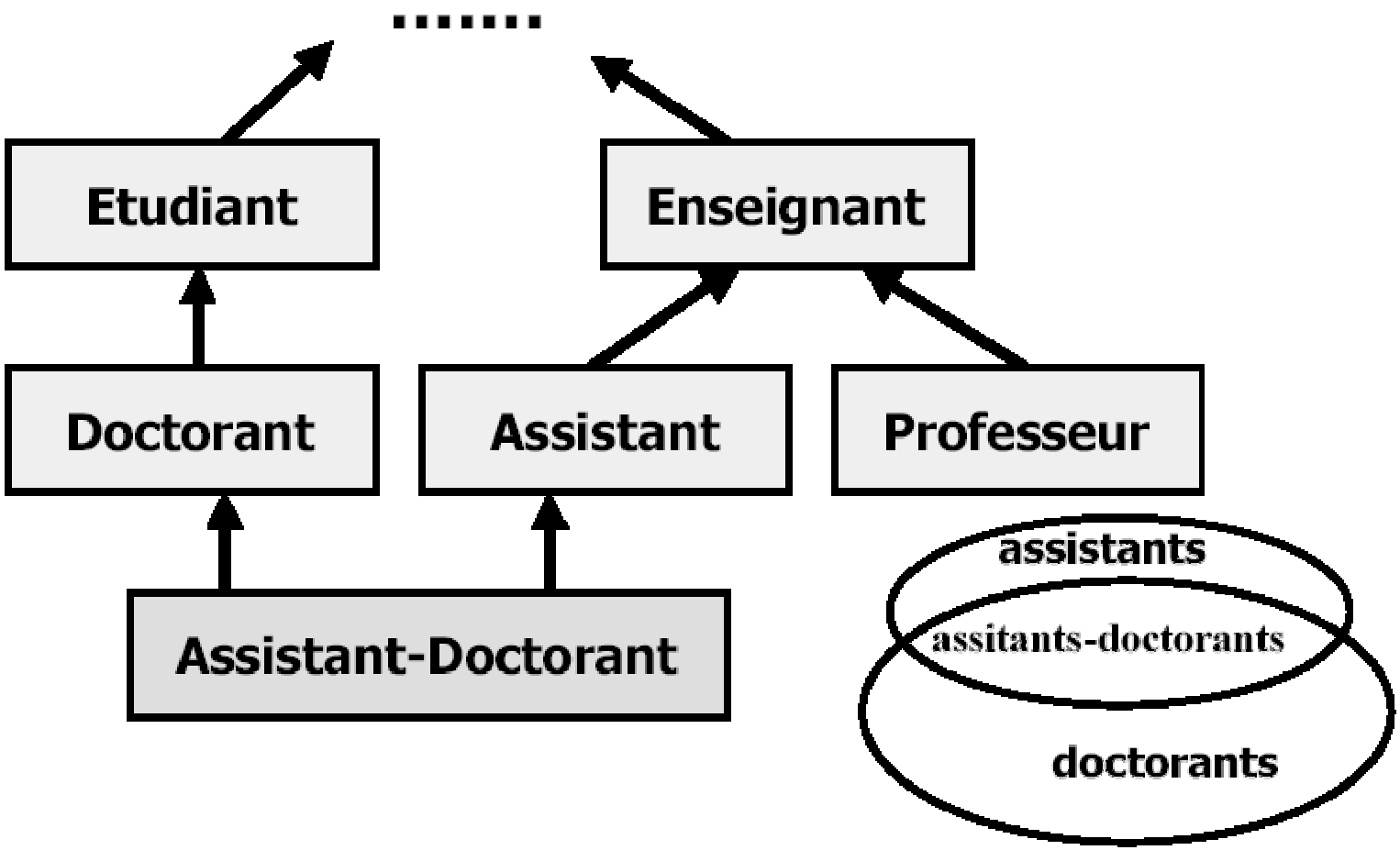
Partition: disjonction + couverture

Clusters de spécialisation

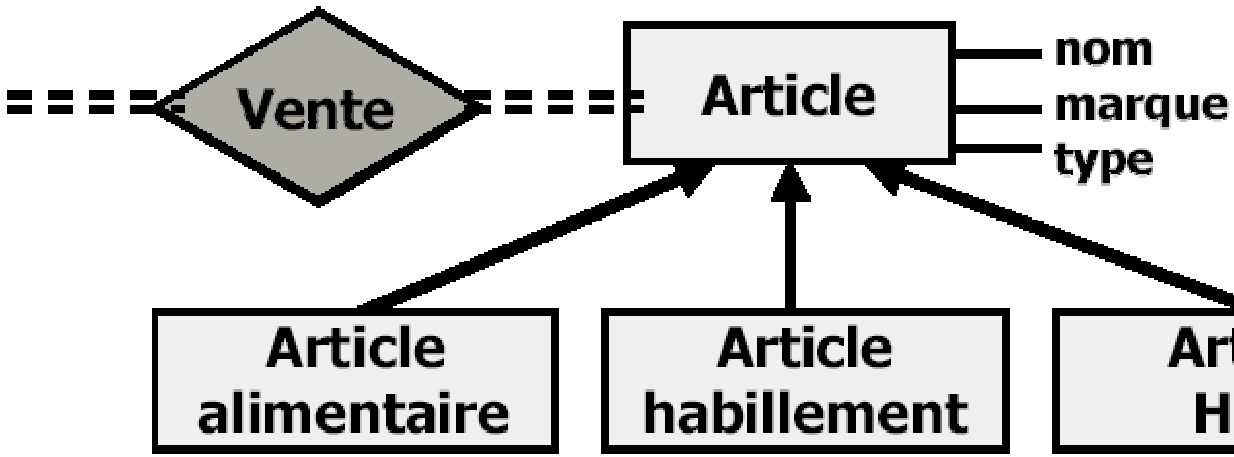


critère: sexe **critère: section** **critère: année**

Généralisation multiple



Héritage



date limite

tailles couleurs

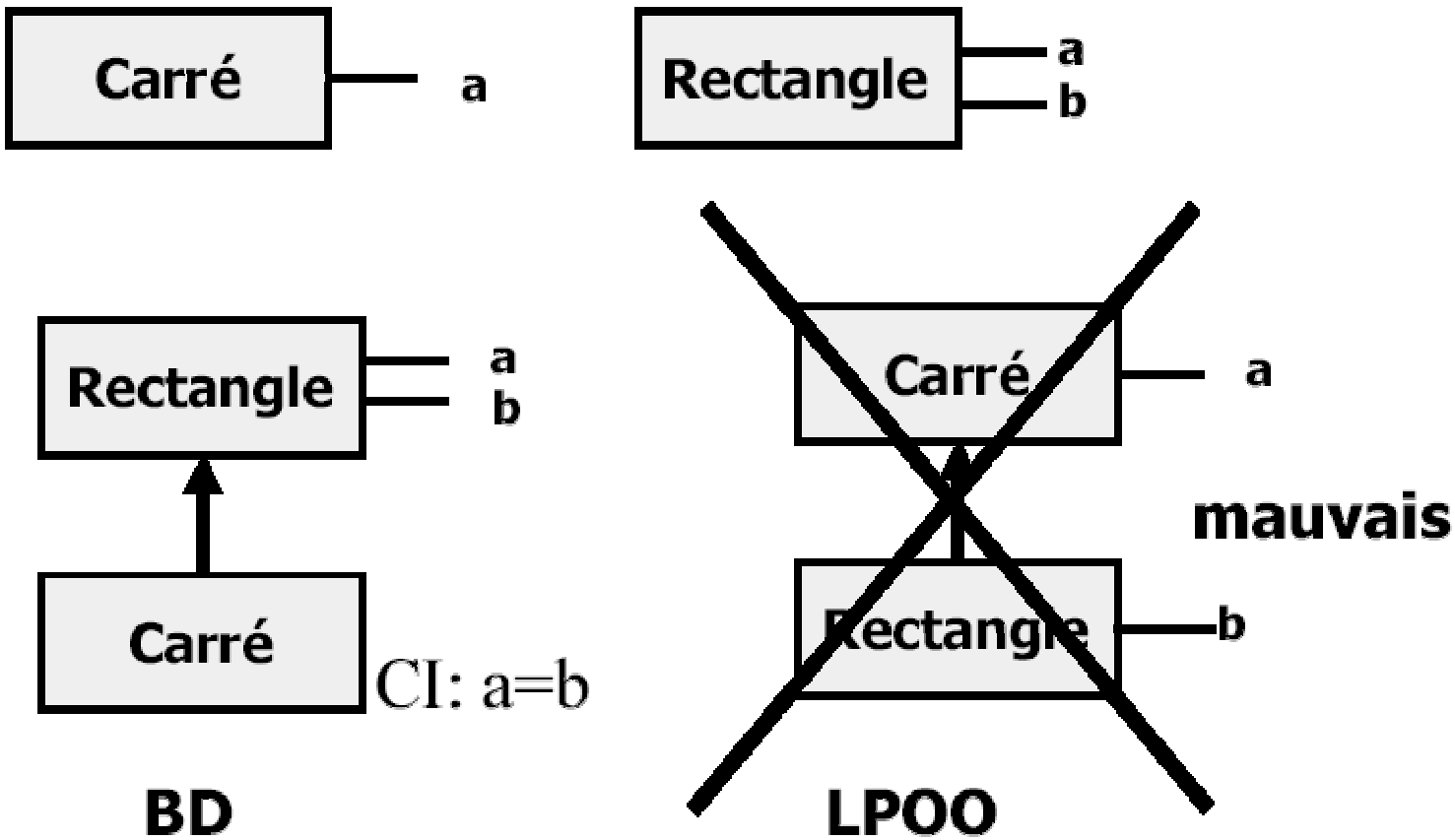
puissance

Reparation

La description des sous-types ne porte que sur les informations additionnelles propres au sous-type

Service après vente

Héritage et inclusion



Description d'un schéma EA

- Types d'entités
- Types d'association
- Attributs
- Liens is-a
- Types d'identifiant
- Domaines d'attribut
- Contraintes d'intégrité

schéma conceptuel

EA = ({TE}, {TA}, {CI})

Description d'un TE

- nom du type d'entité;
- nom du (ou des) type(s) d'entité sur-type de ce type d'entité, s'il en existe;
- une définition libre (commentaire) précisant la sémantique du TE
  - caractérisation exacte de la population du type d'entité
- description des attributs du TE
- composition des identifiants du TE, s'il en existe
- contraintes d'intégrité propres au TE

Les entités dans le temps

TE Client: qu'est qu'un client ???

- Toute personne qui a une commande en cours
- Toute personne qui a fait une commande dans les six derniers mois
- Toute personne qui a fait une commande dans le passé ou qui est susceptible de faire une commande dans le futur
- ...

Description d'un TA

- nom du type d'association
- une définition libre (commentaire) précisant la sémantique du TA
- noms des TE participant au TA, avec le nom du rôle les associant un TA
- pour chaque rôle, sa cardinalité
- description des attributs du TA , s'il en existe
- composition des identifiants du TA, s'il en existe
- contraintes d'intégrité propres au TA

Les associations dans le temps

TA Personne . Emprunte . Livre

- Quels emprunts veut-on dans la BD?
- Seulement les emprunts en cours
- Les emprunts des trois derniers mois
- Aussi les emprunts à venir (réservations)
- …

Exemple: TA "Affecté" (BD hypermarché)

- nom: Affecté
- Définition : "lie un employé au rayon dans lequel cet employé travaille aujourd'hui.
- TE participants: <Employé, > , <Rayon, >
- cardinalités: Employé : min=0, max=1

  Rayon : min=0, max=n
- attributs: /
- identifiant: Employé.nom
- contraintes d'intégrité: /

Description d'un attribut

- nom de l'attribut
- définition libre de sa sémantique
- cardinalités
- si attribut simple: domaine de valeurs
- si attribut complexe: description des attributs composants

Domaine de valeurs d'un attribut simple

- Le domaine de valeurs d'un attribut définit l'ensemble des valeurs permises pour cet attribut.
- Le domaine d'un attribut simple peut être
  - un domaine de base: entiers, réels, string, booléen, date, .
  - un domaine de base avec restriction:

    Entier [>=1,<=12] pour l'attribut mois
  - un domaine énuméré:

    {janvier, février, ., décembre}
  - un domaine de type défini par l.utilisateur

Contraintes d'intégrité (CI)

- règles définissant les états (CI statiques) et les transitions d'état (CI dynamiques) possibles de la BD
- doivent être décrites explicitement (avec un langage approprié) si elles ne peuvent pas être décrites avec les concepts du modèle de données
- une BD est cohérente si toutes les CI définies sont respectées par les valeurs de la BD.

CI sur les attributs

- Restrictions de domaine fixes :
  - âge ∈ [0 : 130]
- Restrictions selon le contexte :
  - SI mois ∈ {4, 6, 9, 11} ALORS jour ∈ [1:30] ,

    SINON SI mois=2 ALORS jour ∈ [1:29],

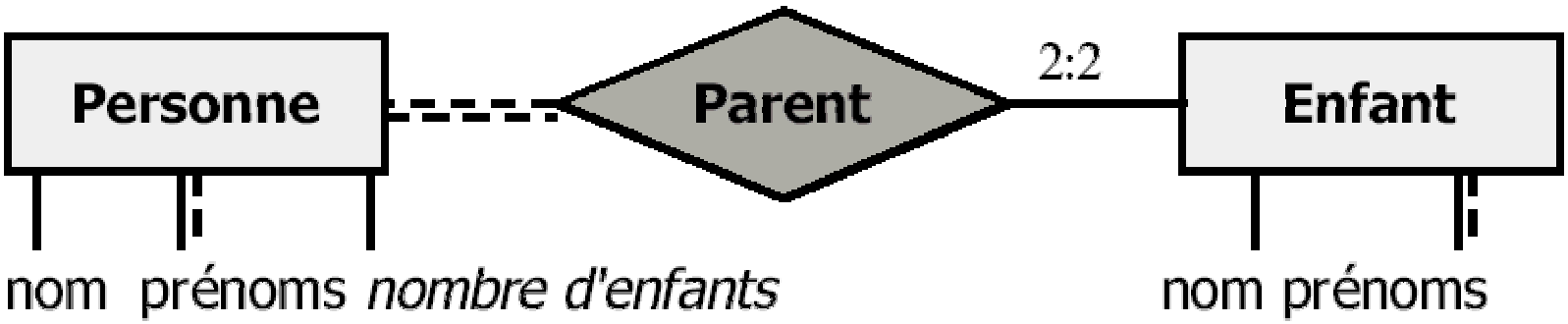
    SINON jour ∈ [1:31]
  - ∀ x,y ∈ Personne, <x,y> ∈ Mariage

    => x.état-civil = "marié" & y.état-civil = "marié.
  - ∀ x ∈ Personne, ∀ y ∈ Voiture, <x,y> ∈ Conduit

    => x.âge >=18
  - ∀ x ∈ Personne,

    (x.sexe=F OR x.age<18) => x.statut_milit.=NUL

Attributs dérivés



nombre d'enfants =

nombre d'occurrences du TA .Parent. Qui lient cette Personne

Chapitre 3 bis

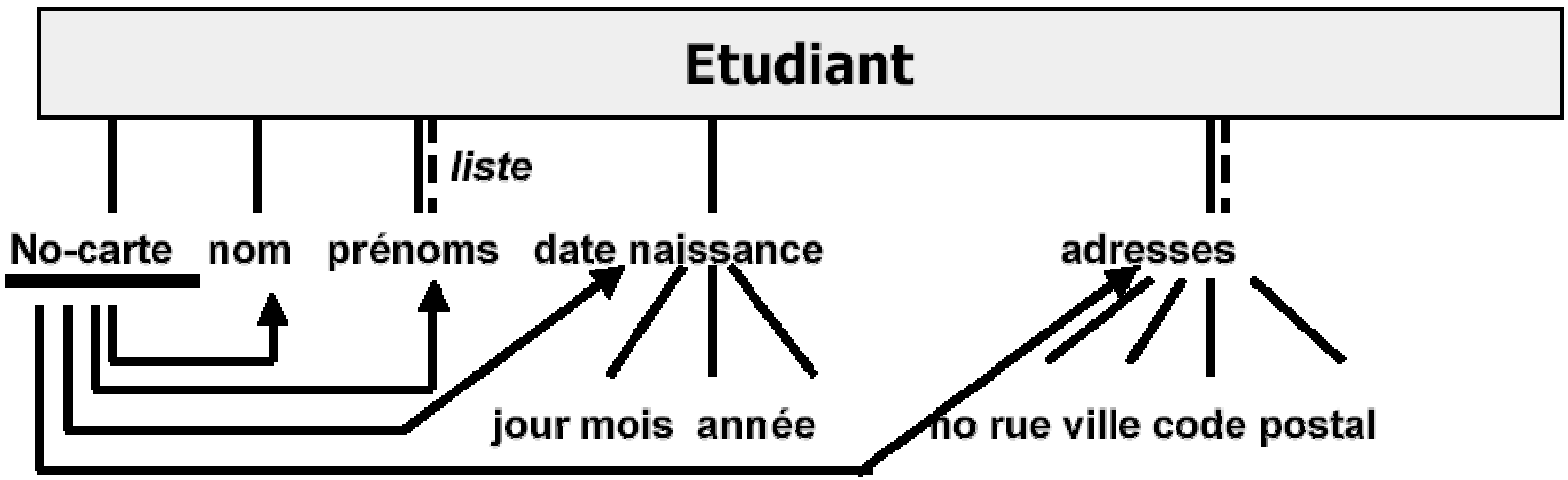
Modélisation Conceptuelle : Validation et Transformations

Validation d'un schéma EA

- Syntaxique: respect des règles du modèle
- Par confrontation aux dépendances:
  - règles de normalisation
- Par jeu d'essai
- Complétude par rapport aux traitements
- Par les utilisateurs

Règles à connaître et à appliquer !!!

Concept de dépendance

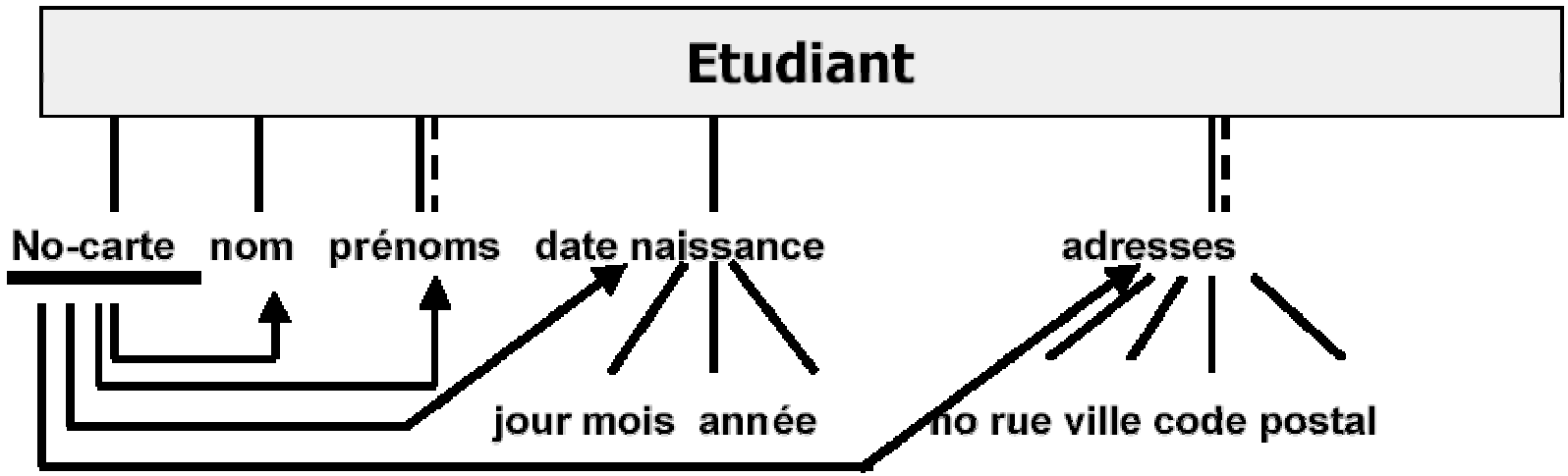


N°carte ⇨ nom, prénoms, date naissance, adresses

A⇨ B si le fait que deux occurrences aient la même valeur pour A entraîne nécessairement qu'elles aient la même valeur pour B.

A ⇨ B : «B dépend de A», «A déterminеB »

Validation d'un TE(TA) / dépendances



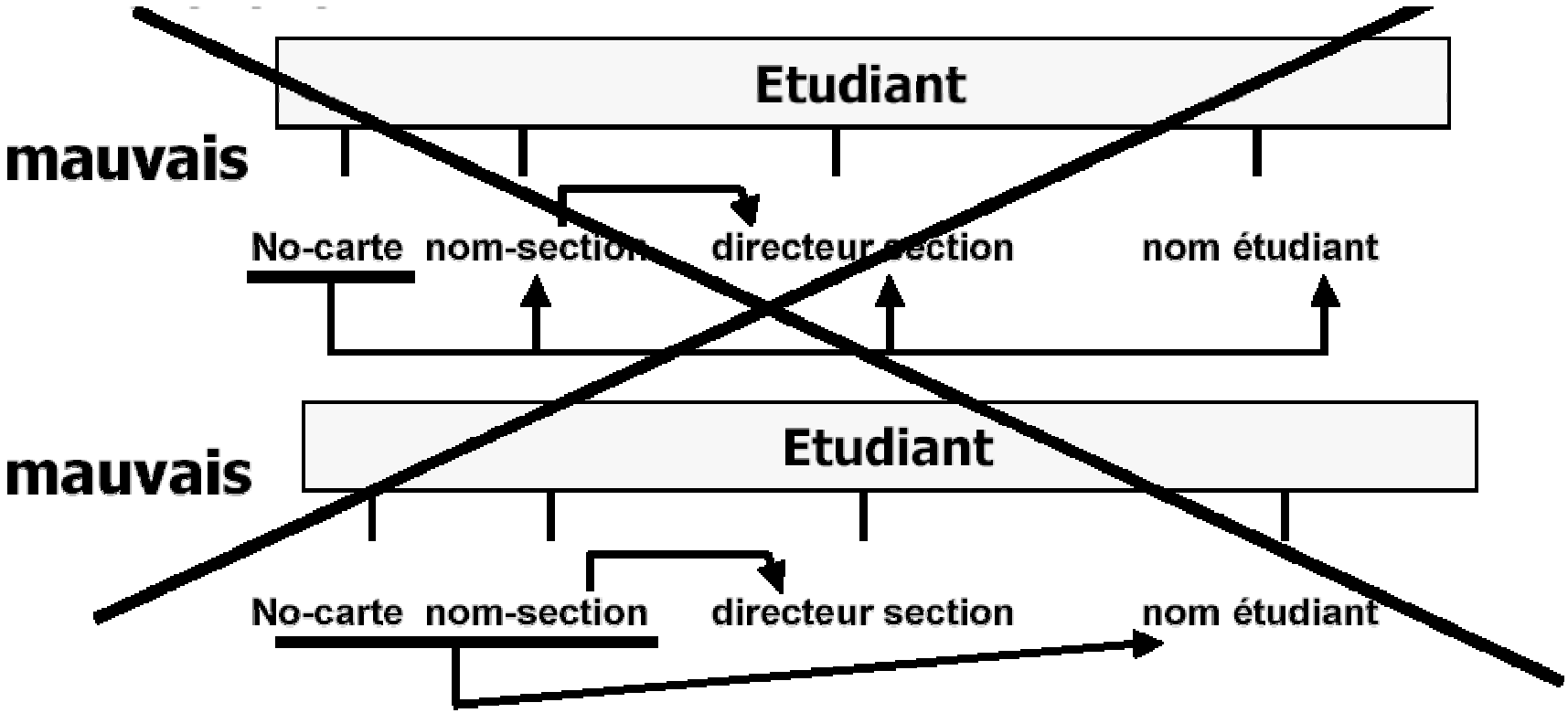
Règle 1: dans un TE (TA) valide, tous les attributs directs (simples et complexes) dépendent uniquement de chaque identifiant entier du TE (TA).

n°carte, nom, prénoms, date naissance et adresses **sont les attributs directs d'*Etudiant*, qui a pour identifiant *n°carte***

Schémas incorrects

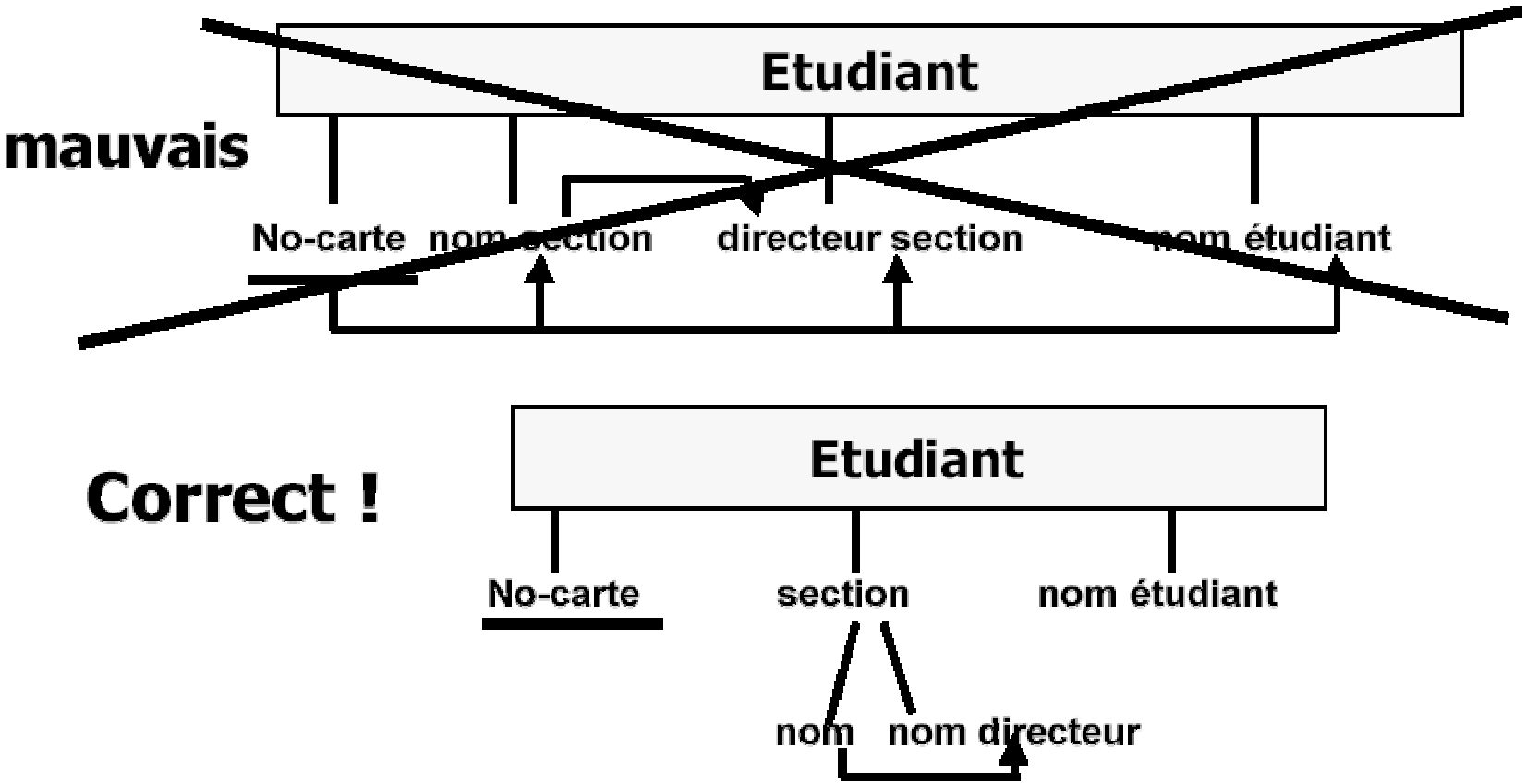
La règle est contredite si un attribut dépend d'un

partie de l'identifiant ou d'un autre attribut non identifiant.



Normalisation

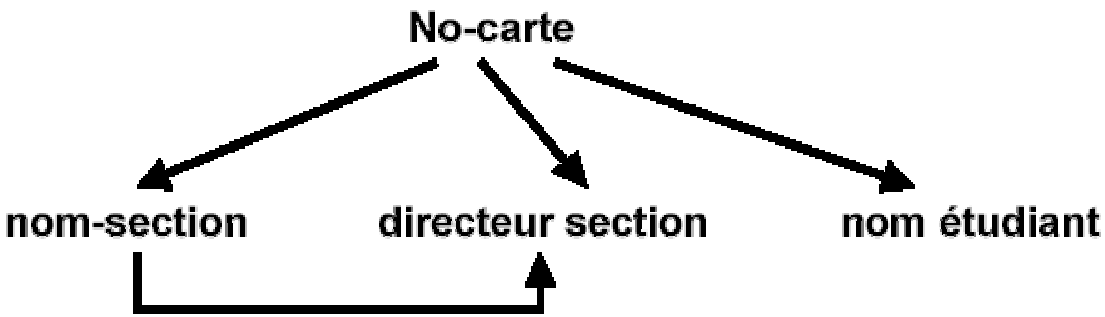
Processus de modification d'un schéma qui conduit à obtenir un schéma offrant les propriétés désirées.



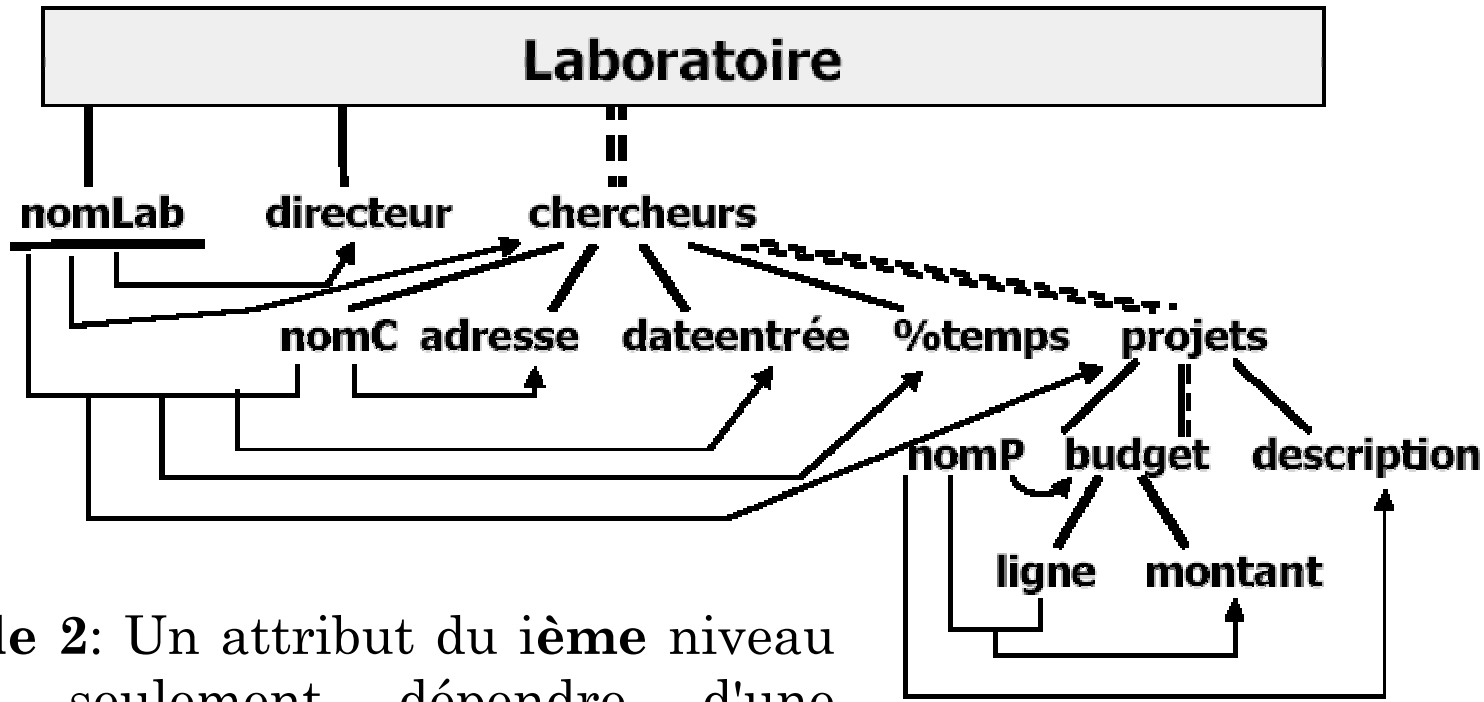
Dépendance et identifiant

- Graphe des dépendances

- L'identifiant est la racine du graphe

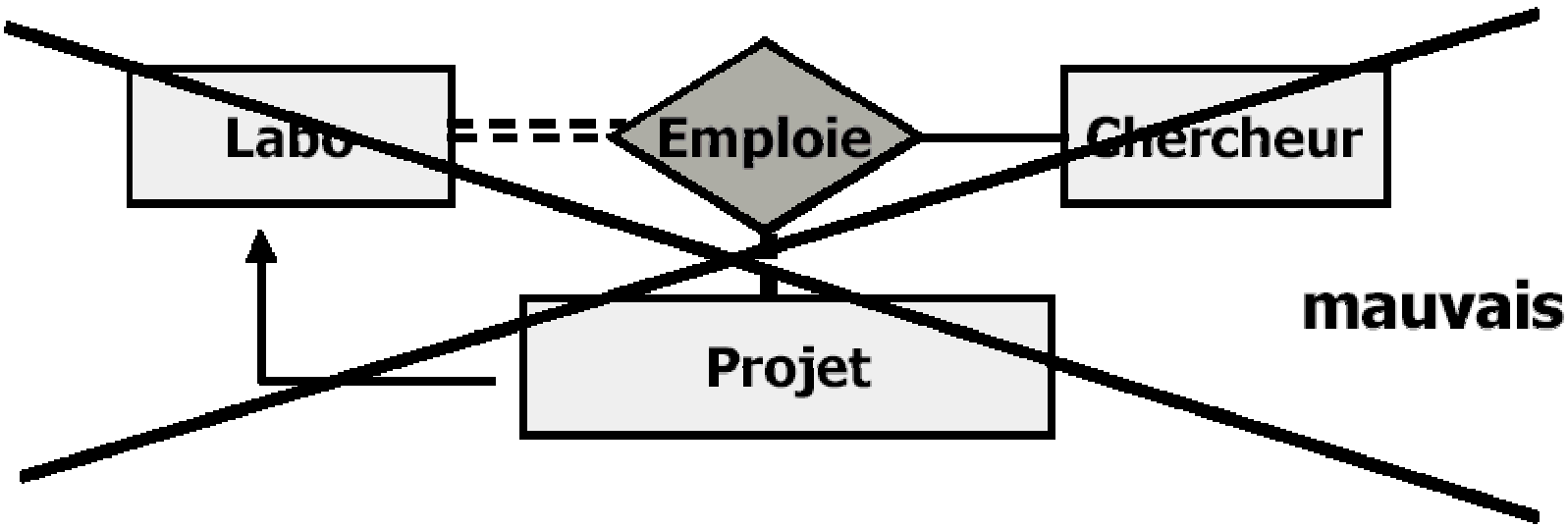


Validation / attributs complexes



Règle 2: Un attribut du i**ème** niveau peut seulement dépendre d'une combinaison d'attributs du même niveau et de niveaux supérieurs contigus.

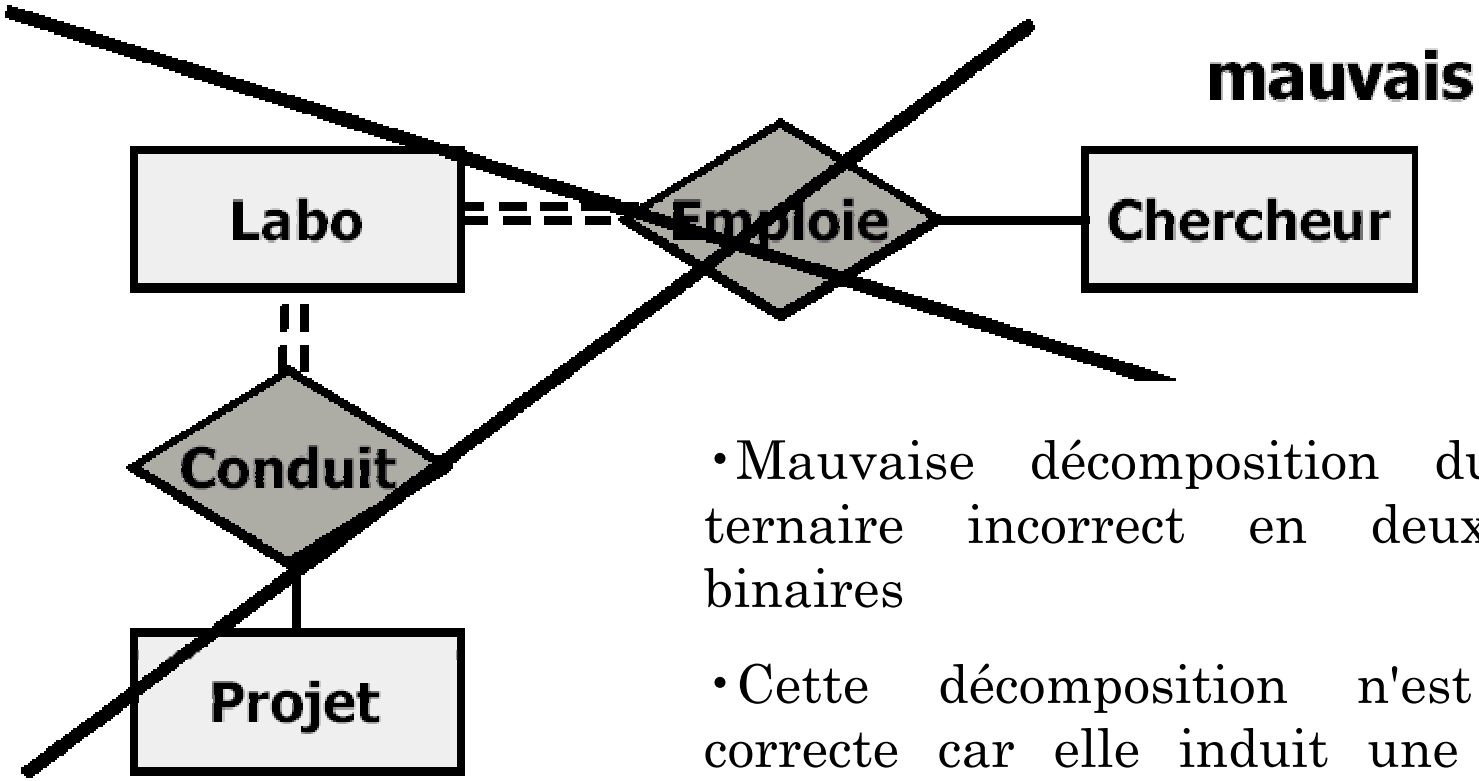
Dépendances entre TE



- Si tout projet n'est fait que par un seul labo, le schéma est incorrect

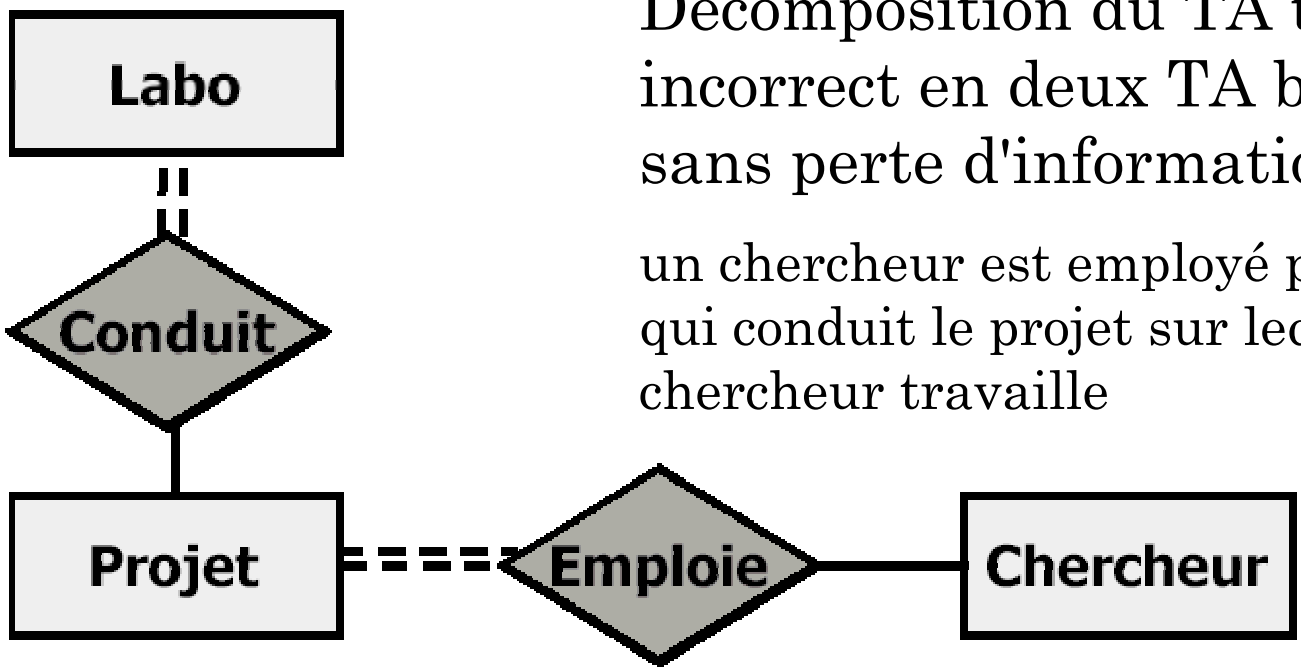
- Règle 3: un TA n-aire (n>2) avec une dépendance entre ses TE doit être decomposé

Normalisation du TA: incorrect



- Mauvaise décomposition du TA ternaire incorrect en deux TA binaires
- Cette décomposition n'est pas correcte car elle induit une perte d'information – on ne sait plus sur quel projet travaille un chercheur !!

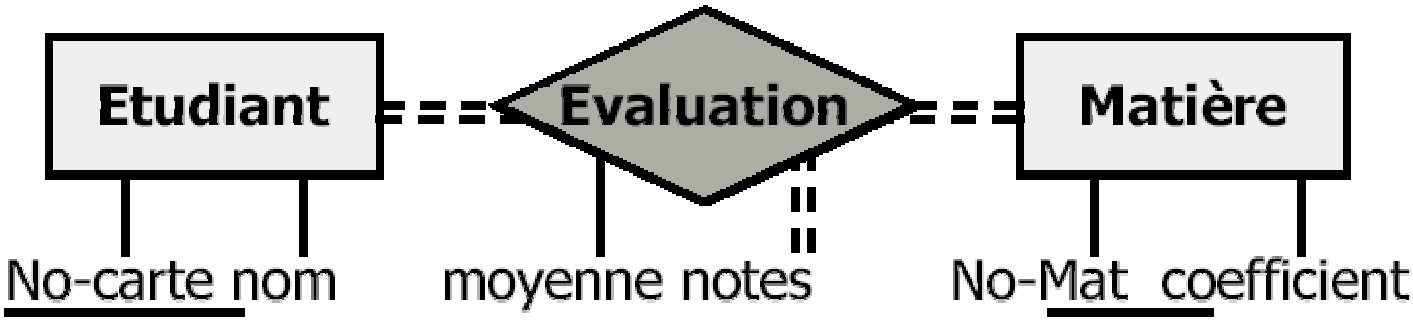
Normalisation du TA: correct



Décomposition du TA ternaire incorrect en deux TA binaires sans perte d'information:

un chercheur est employé par le labo qui conduit le projet sur lequel le chercheur travaille

Validation des attributs d'un TA

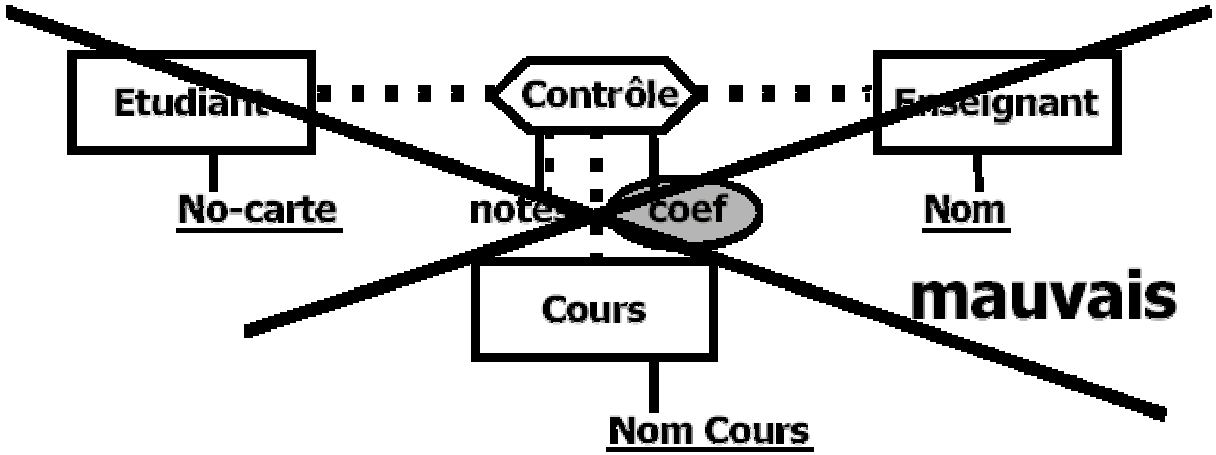


Règle 4: dans un TA sans dépendance entre les TEs liés, les attributs du TA dépendent de tous les TE liés par ce TA.

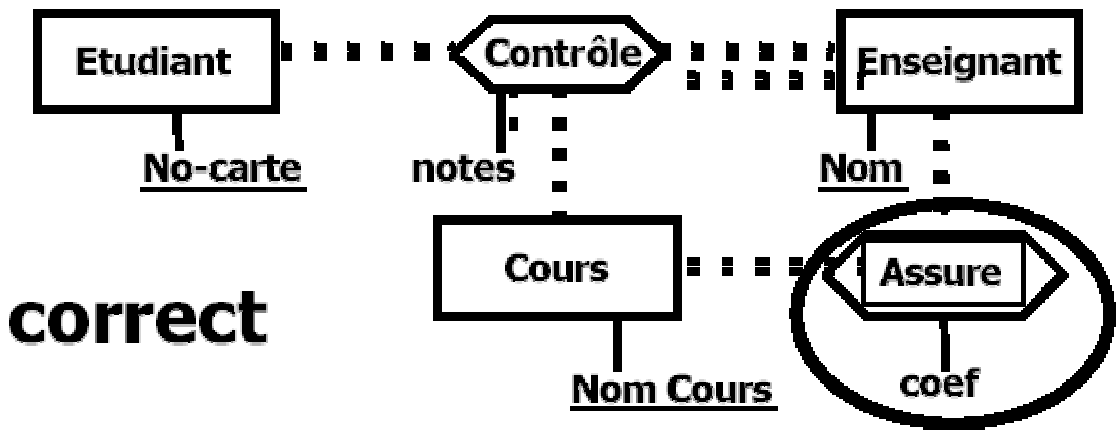
(No-carte,No-Mat) ⇨ moyenne, notes

Validation des attributs d.un TA

Si Coef = fonction du nombre d'heures assurées par l'enseignant dans ce cours.



Alors Coef ne dépend pas D'Etudiant



Elimination des TA redondants

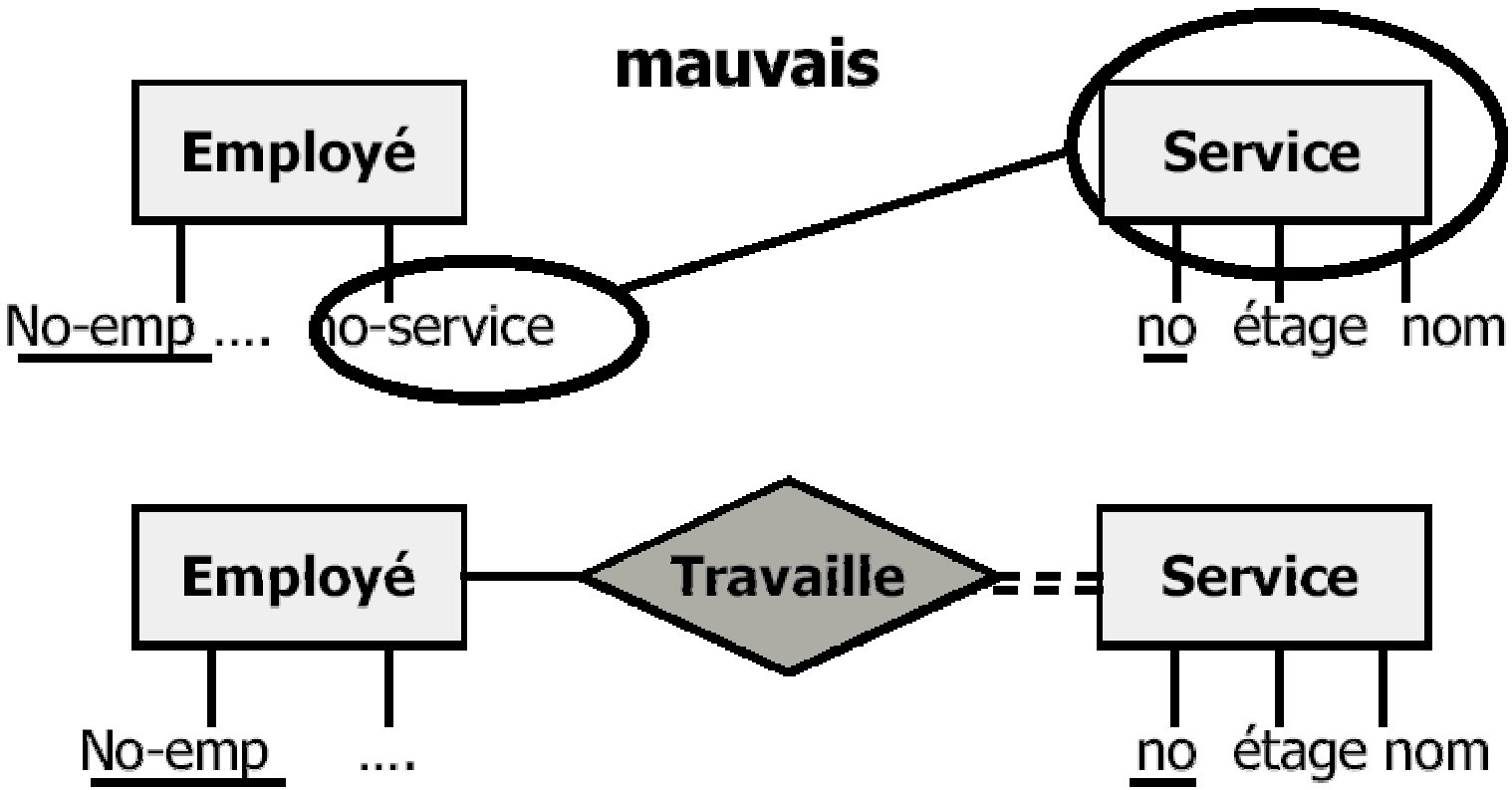
Si

"Est élève de" = Inscrit-Cours-Assure

alors il y a redondance inutile.

On supprime "Est élève de".

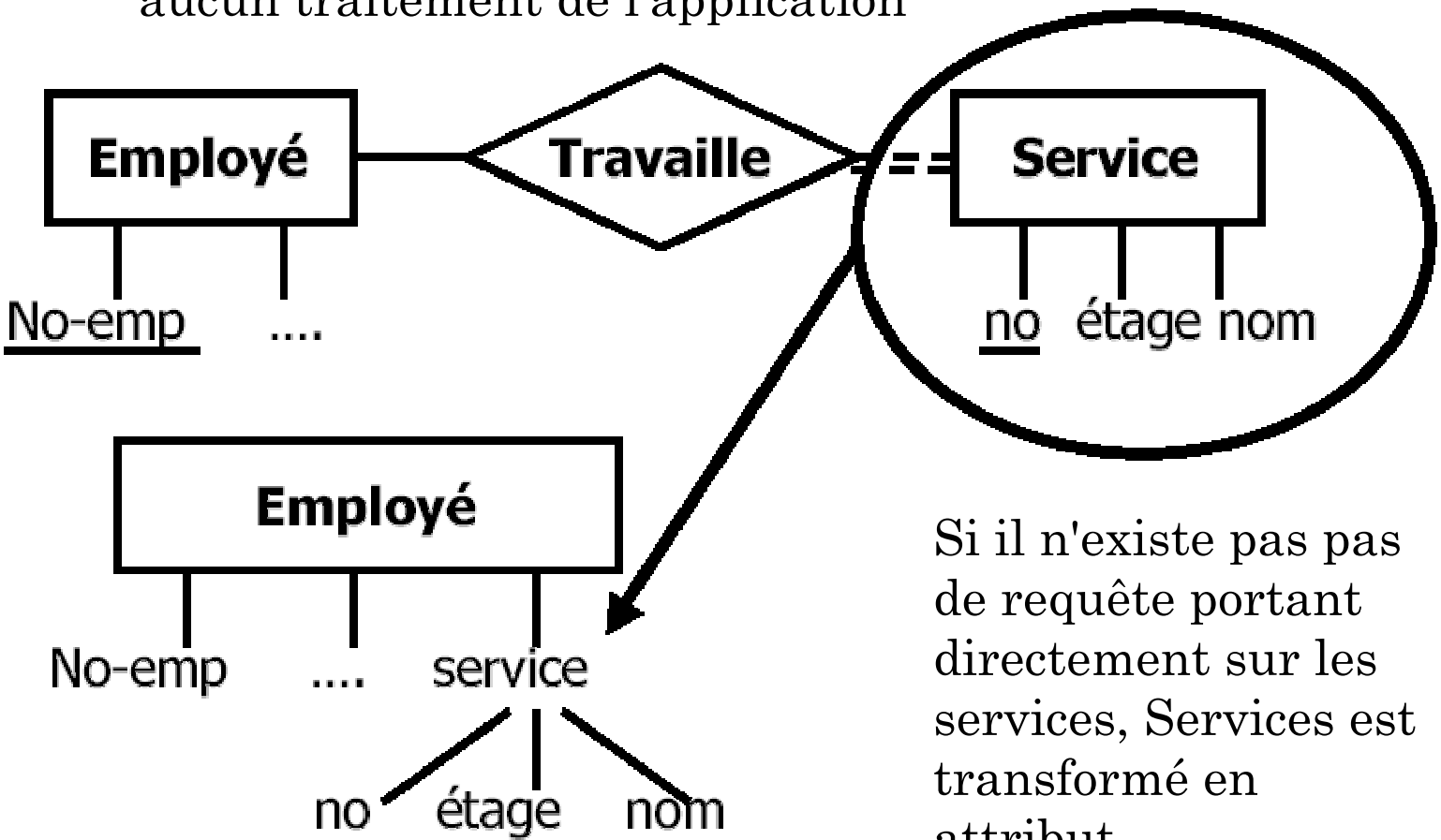
Remplacement d'un attribut par un TA



Règle de remplacement

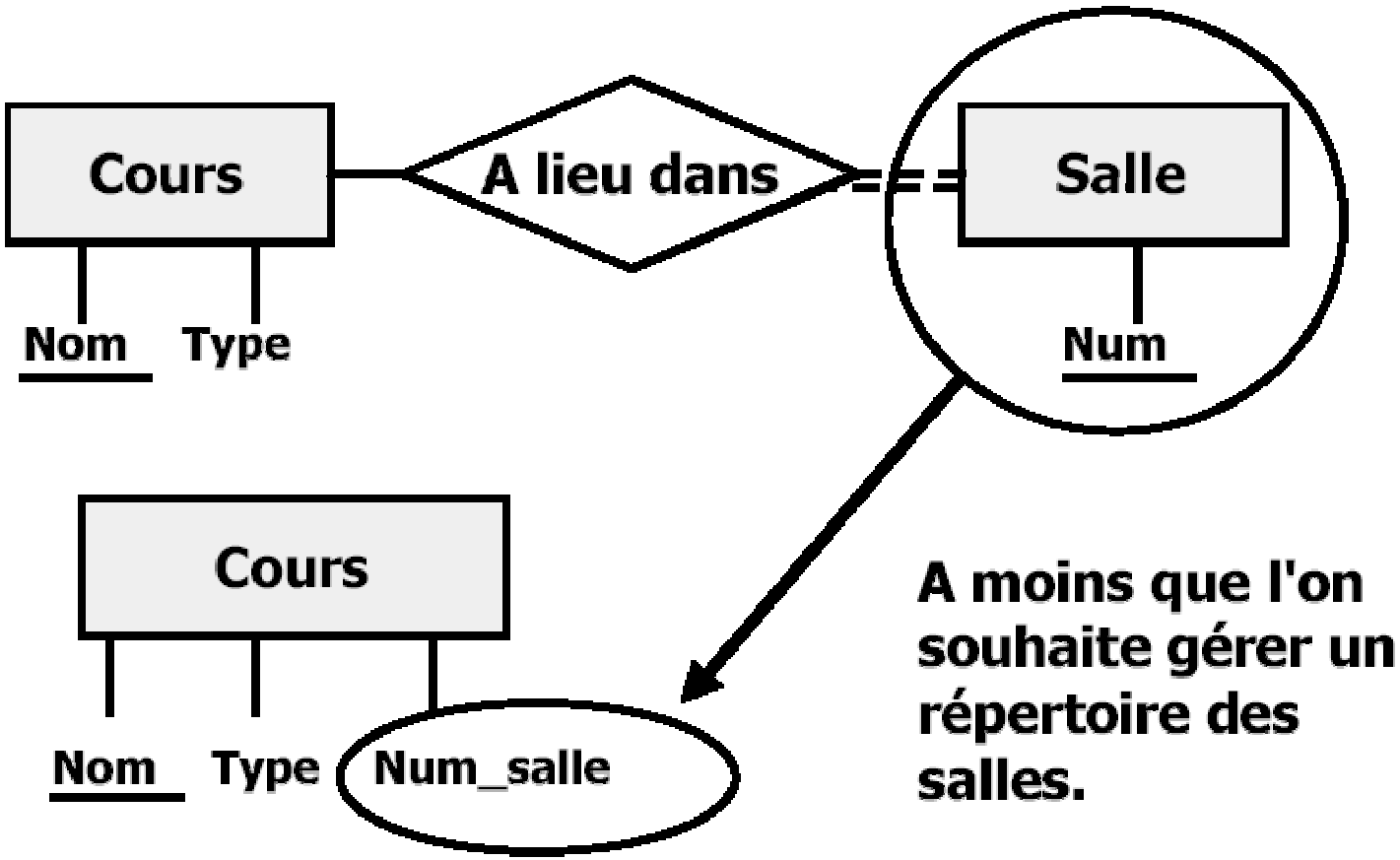
Elimination des TE inutiles

Un TE est inutile s'il ne présente d'intérêt pour aucun traitement de l'application



Si il n'existe pas pas de requête portant directement sur les services, Services est transformé en attribut.

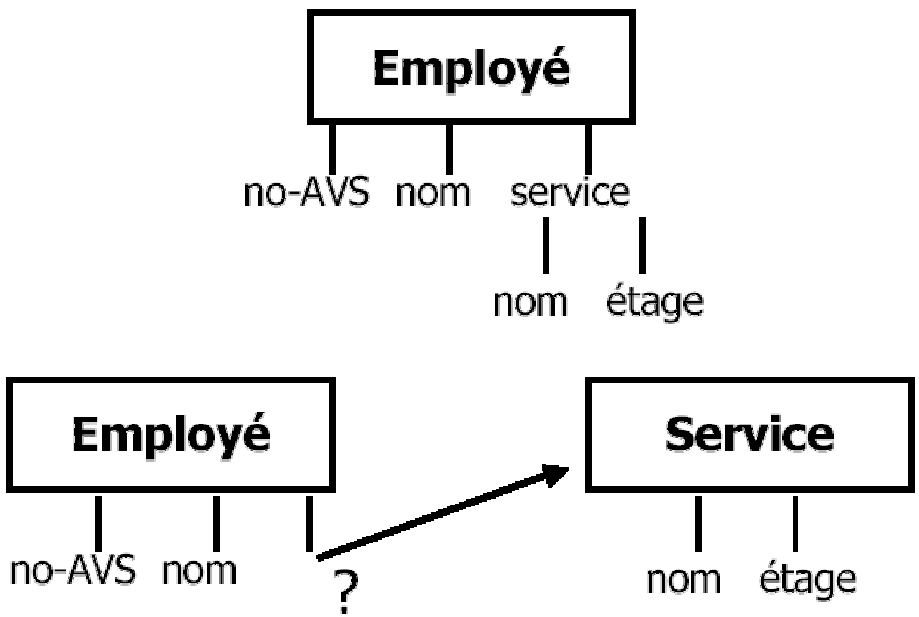
TE répertoires ou attributs ?



Choix de modélisation

- TE ou attribut ?
- TE ou TA ?
- TA ou attribut ?
- Types génériques ou types spécialisés ?
- Attribut optionnel ou sous-type ?

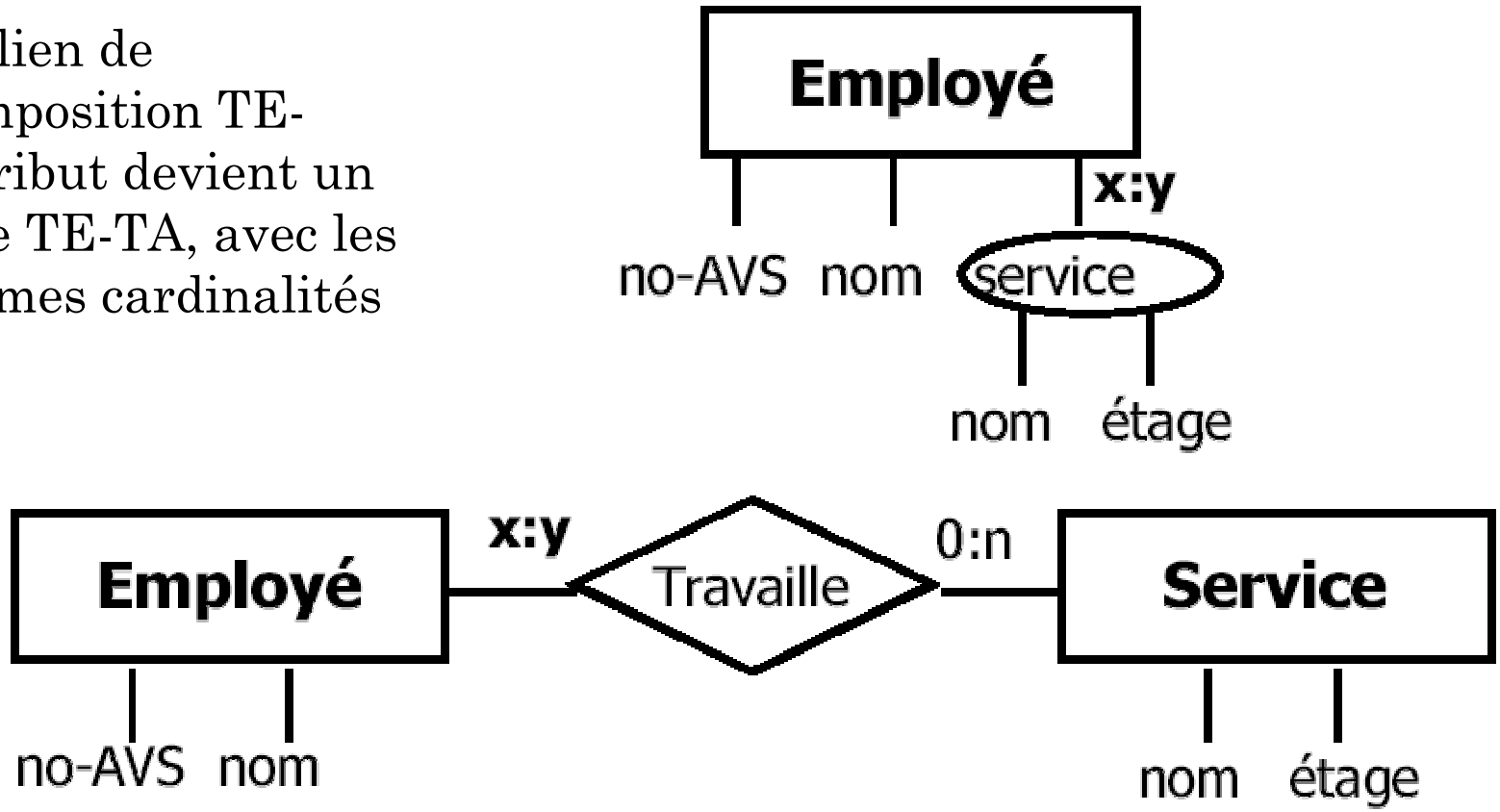
TE ou attribut ?



Transformation d'attribut en TE

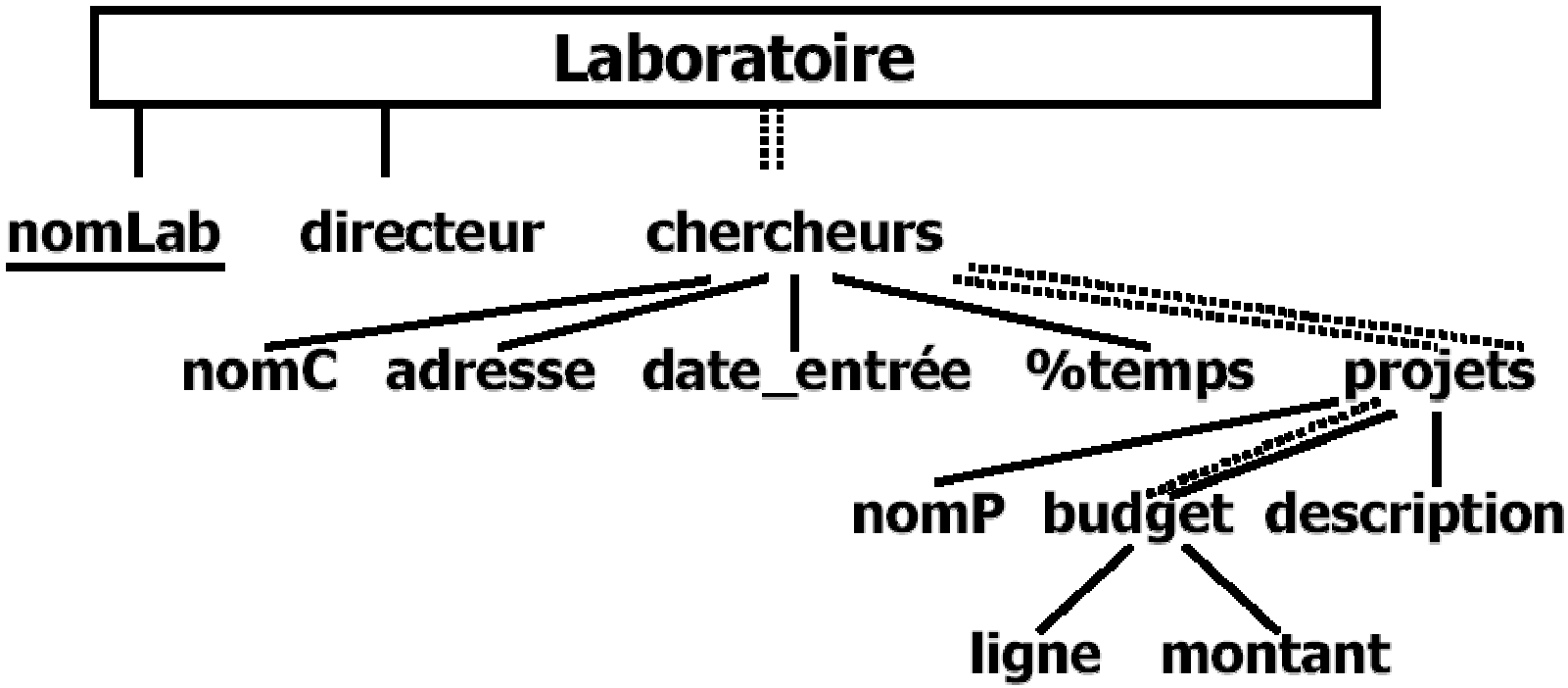
Attribut direct

Le lien de composition TE-attribut devient un rôle TE-TA, avec les mêmes cardinalités

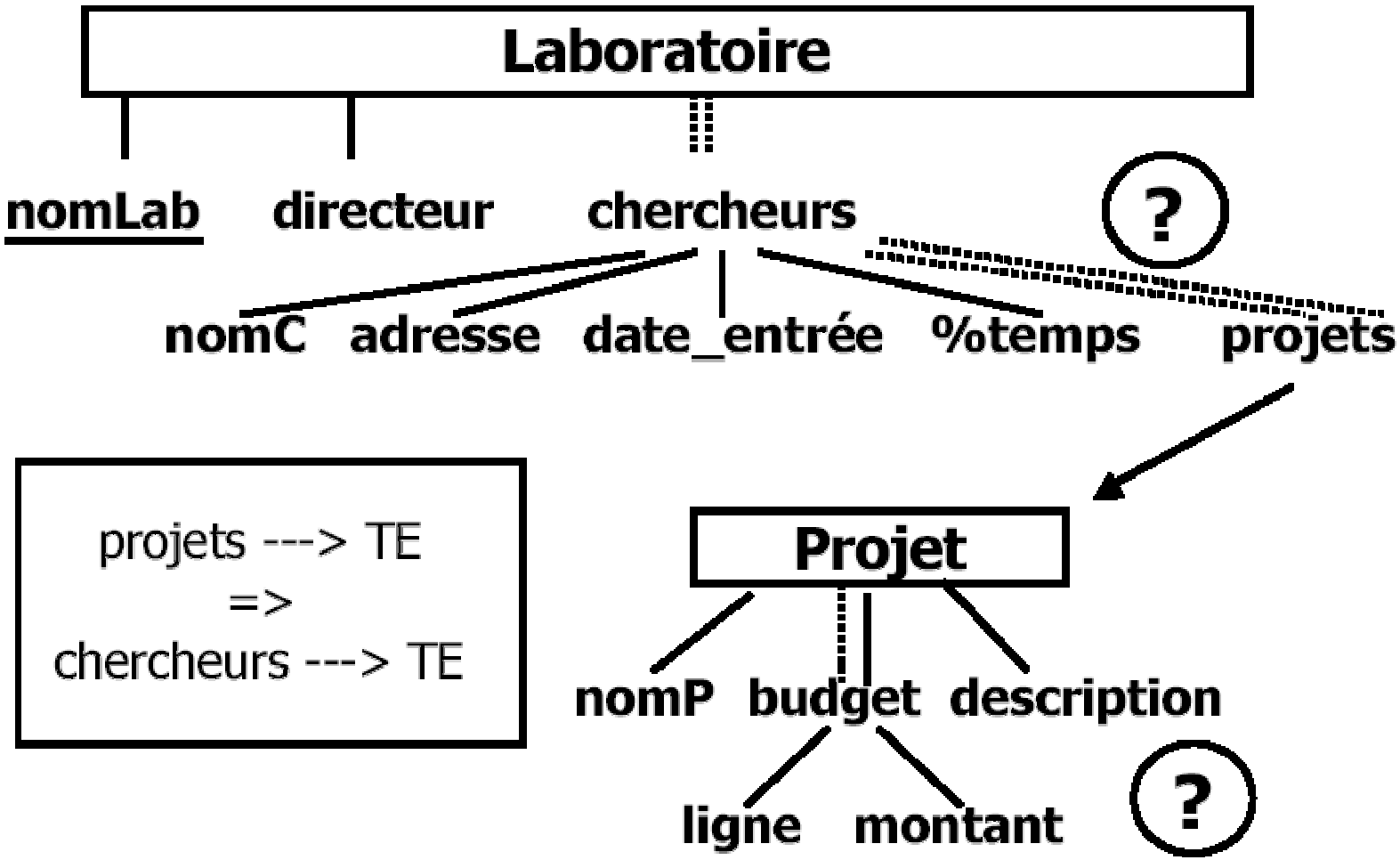


Transformation d'attribut en TE

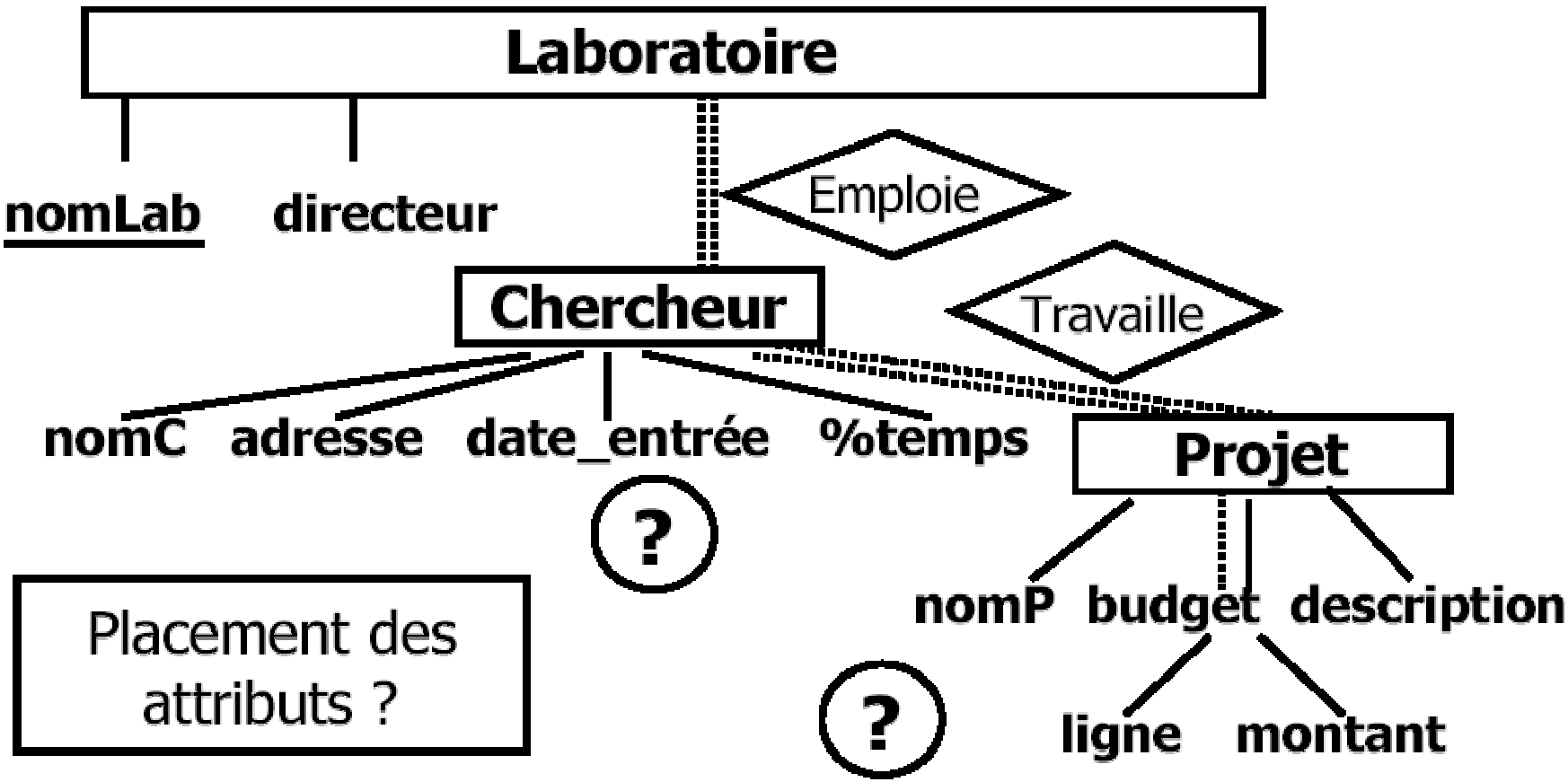
Attribut indirect



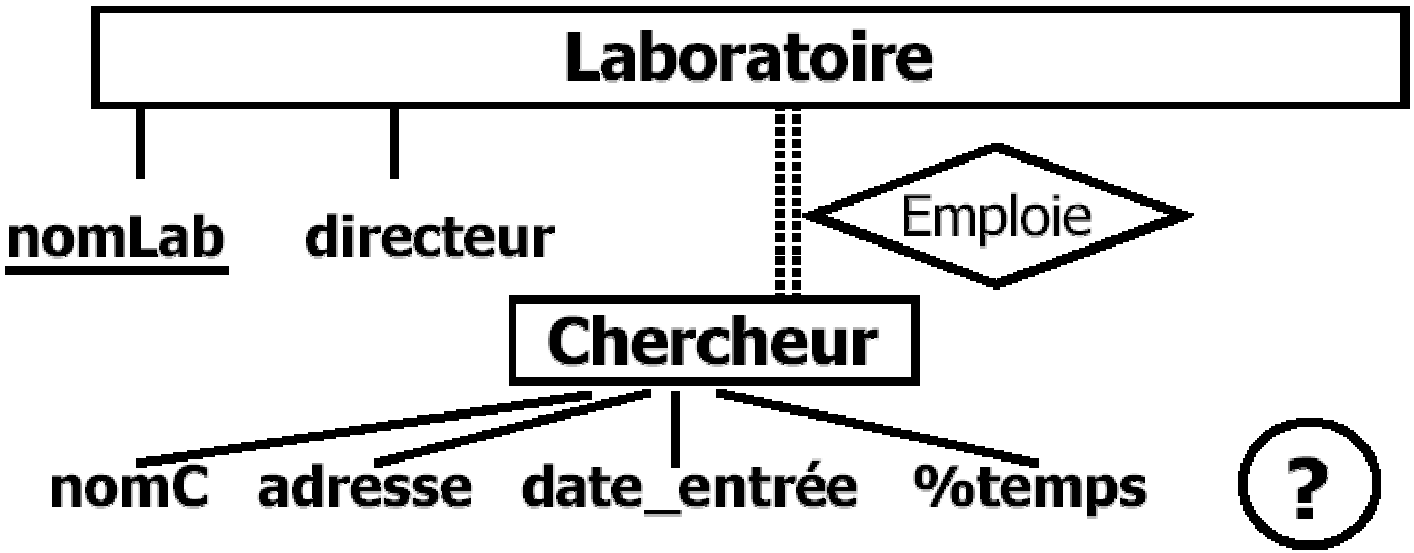
Attribut ⇨ TE: 1ère étape



Attribut ⇨ TE: 2ème étape



Attribut ⇨ TE: 3ème étape



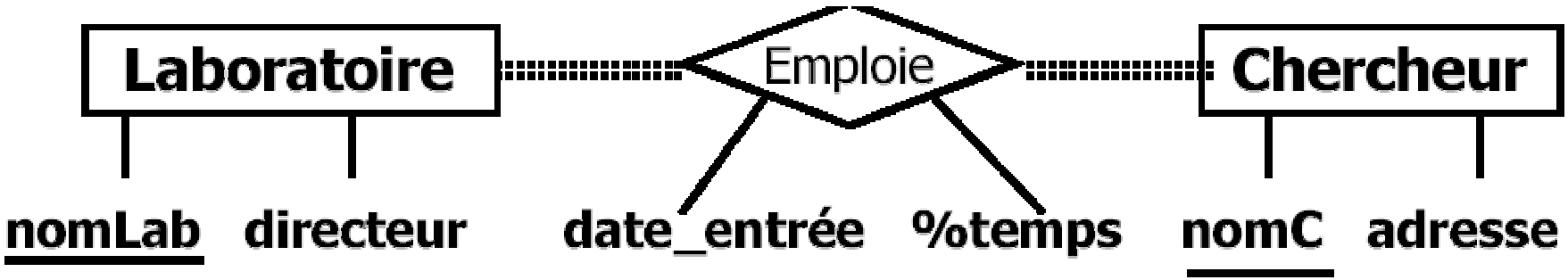
Chercheur ➔ nomC, adresse

⇨ nomC, adresse attributs de Chercheur

(Chercheur,Laboratoire) ➔ date_entrée,%temps

⇨ date_entrée,%temps attributs de Emploie

Attribut ⇨ TE: 3ème étape

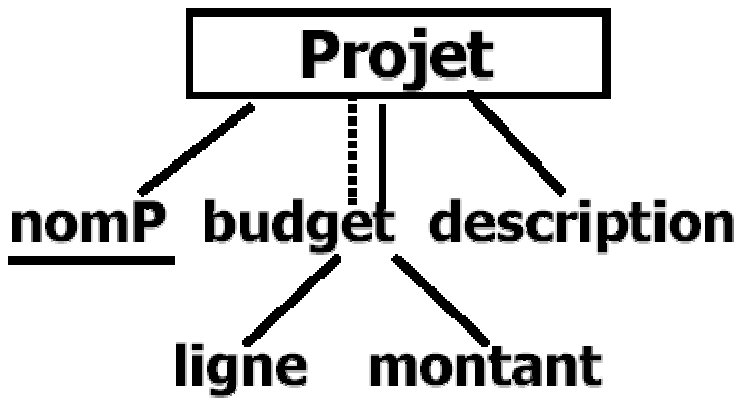


nomC ➔ adresse : nomC identifiant de Chercheur

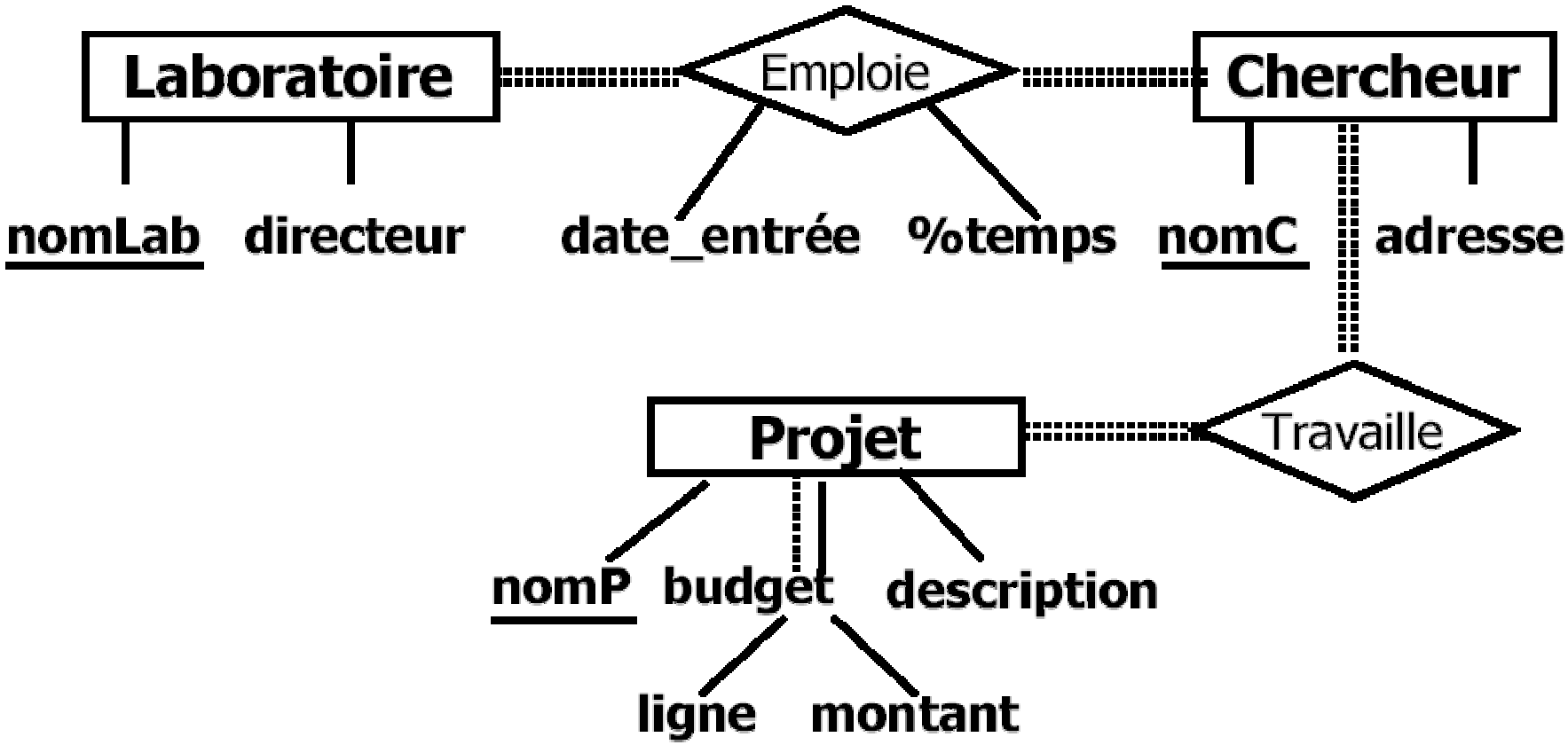
Projet ➔ nomP, budget, description

⇨ attributs de Projet

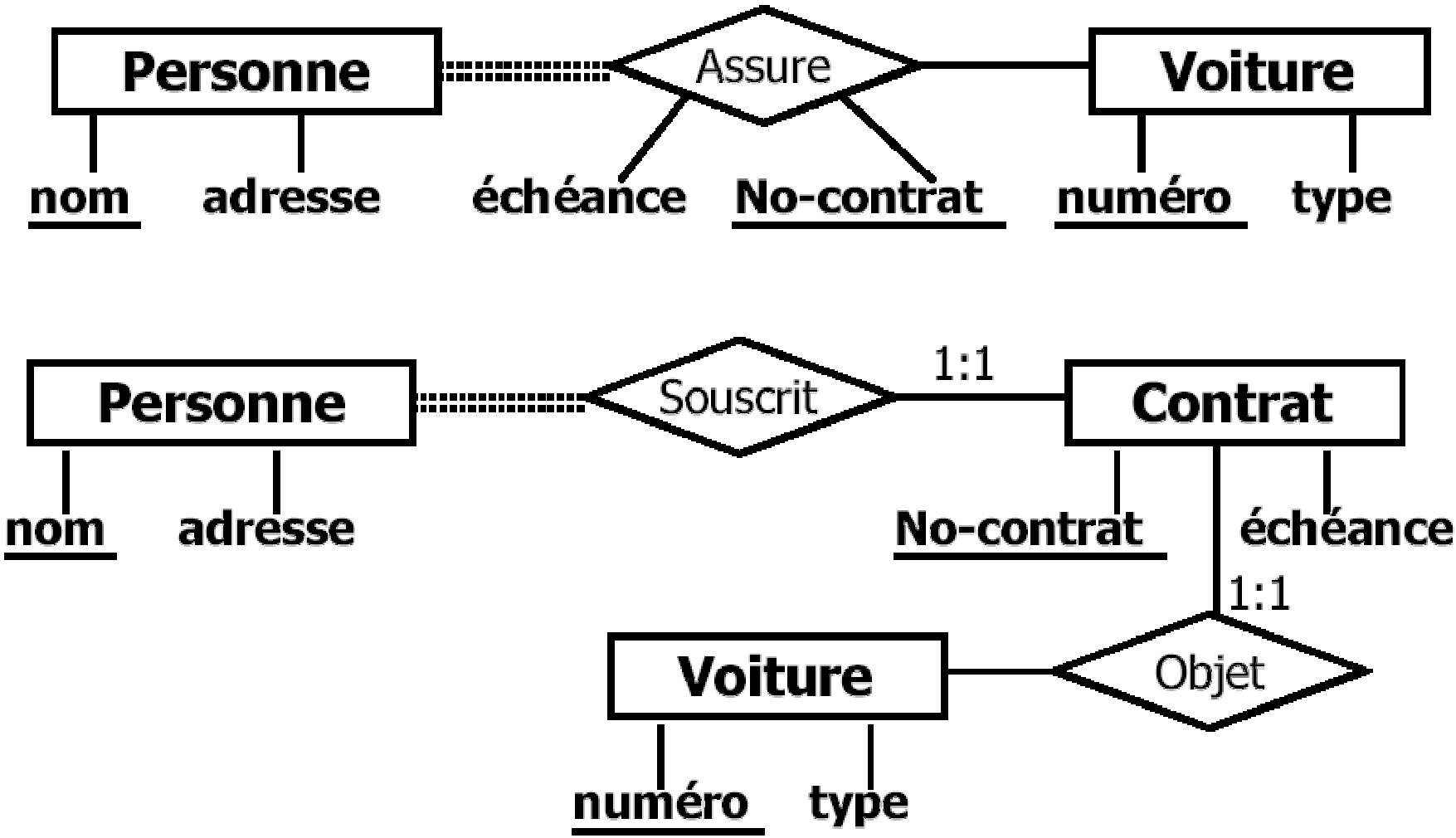
nomP ➔ budget, description



Attribut ⇨ TE: résultat

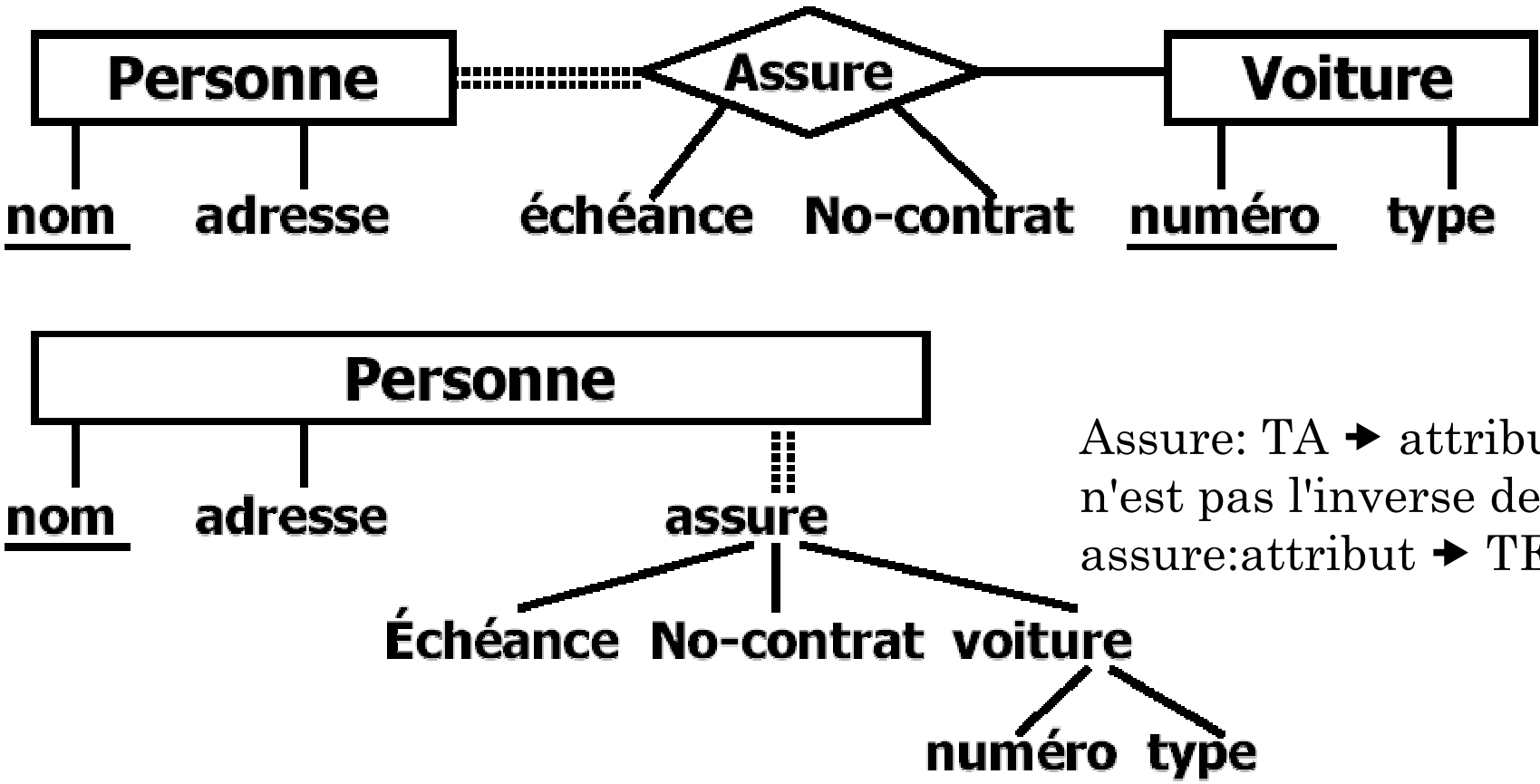


TE ou TA: Conversion TA ⇨ TE



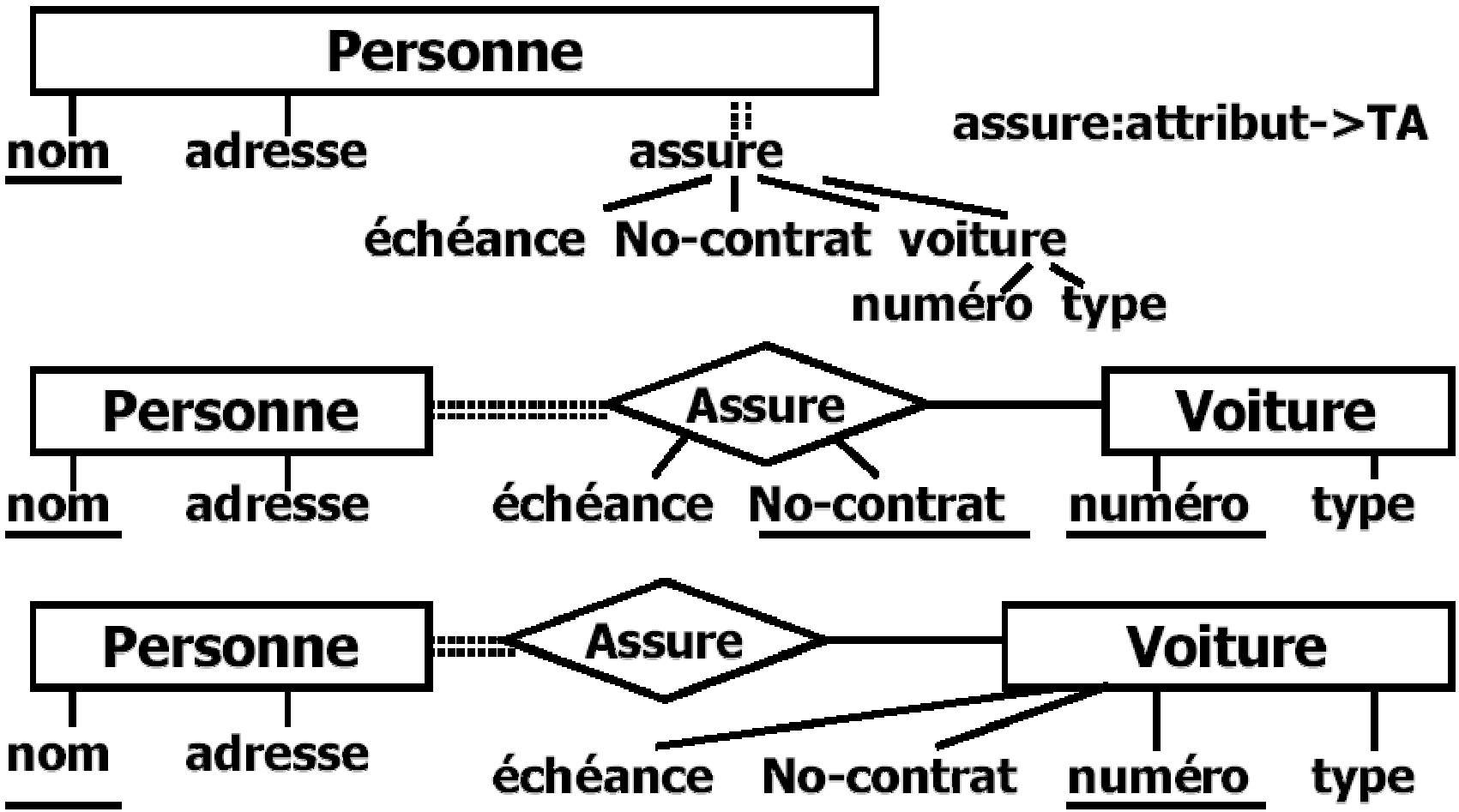
TA ou attribut

- Similaire TE ou attribut

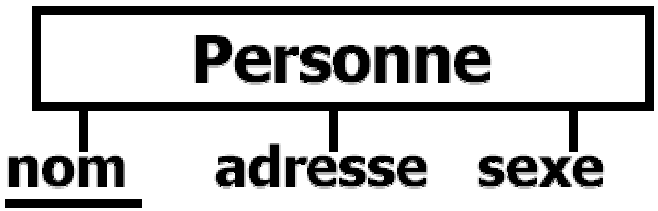


Assure: TA → attribut
n'est pas l'inverse de
assure:attribut → TE

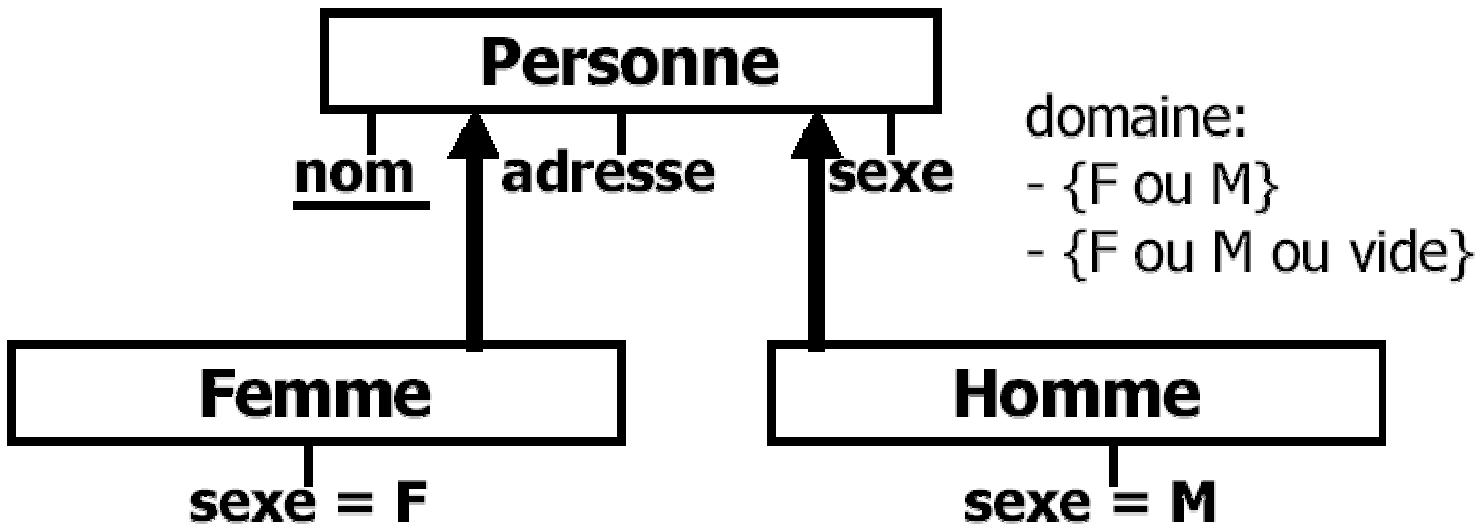
Attribut de TA ou attribut de TE ?



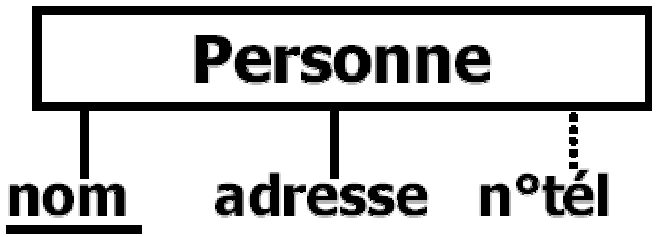
TE génériques/spécifiques



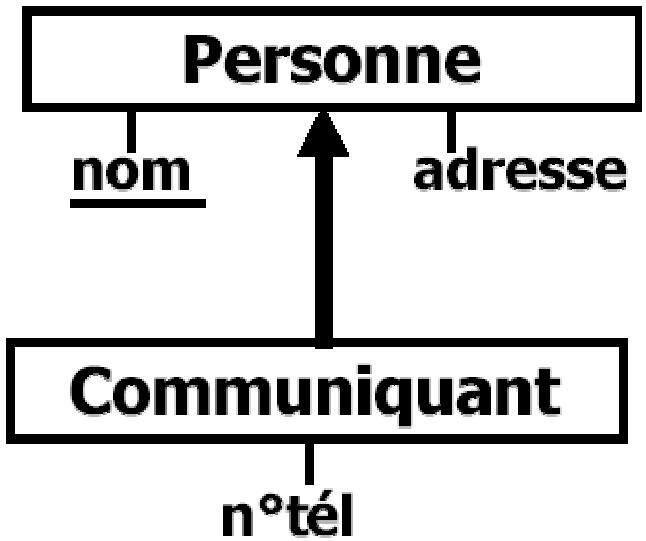
- ou



Attribut optionnel ou sous-type



- ou



Conclusion

- Les transformations de schéma à semantique équivalente (i.e., sans perte d'information) sont un outil puissant de flexibilité
- Elles permettent d'offrir des vues différentes (personnalisées) sur un même contenu informatif
- Elles permettent de passer d'une structure obéissant à certaines règles à une autre structure équivalente obéissant à d'autres règles (exemple: traduction d'un schéma EA en schéma relationnel)

Chapitre 4

Le Modèle Relationnel

Le modèle relationnel

- modèle de niveau logique, très simple
- défini par Ted Codd en 1970; prix Turing en 1986. Développé par IBM lab.
- aujourd'hui utilisé par beaucoup de SGBD commerciaux (Oracle, Informix, DB2, Ingres, Sybase, dBase, Access …) et GIS
- modèle à deux concepts:
  - relation (table)
  - attribut (colonne)

Concepts de base

schéma

Etudiant			
N°Etud	**nom**	**prénom**	**age**

population

136	Dupont	Jean	19
253	Aubry	Annie	20
101	Duval	André	21
147	Dupont	Marc	21

tuple ou occurrence

.

Schéma relationnel

- une BD = ensemble de relations
- schéma d'une BD relationnelle = un ensemble de schémas de relation: R1, R2, …, Rx
- schéma d'une relation = un ensemble d'attributs

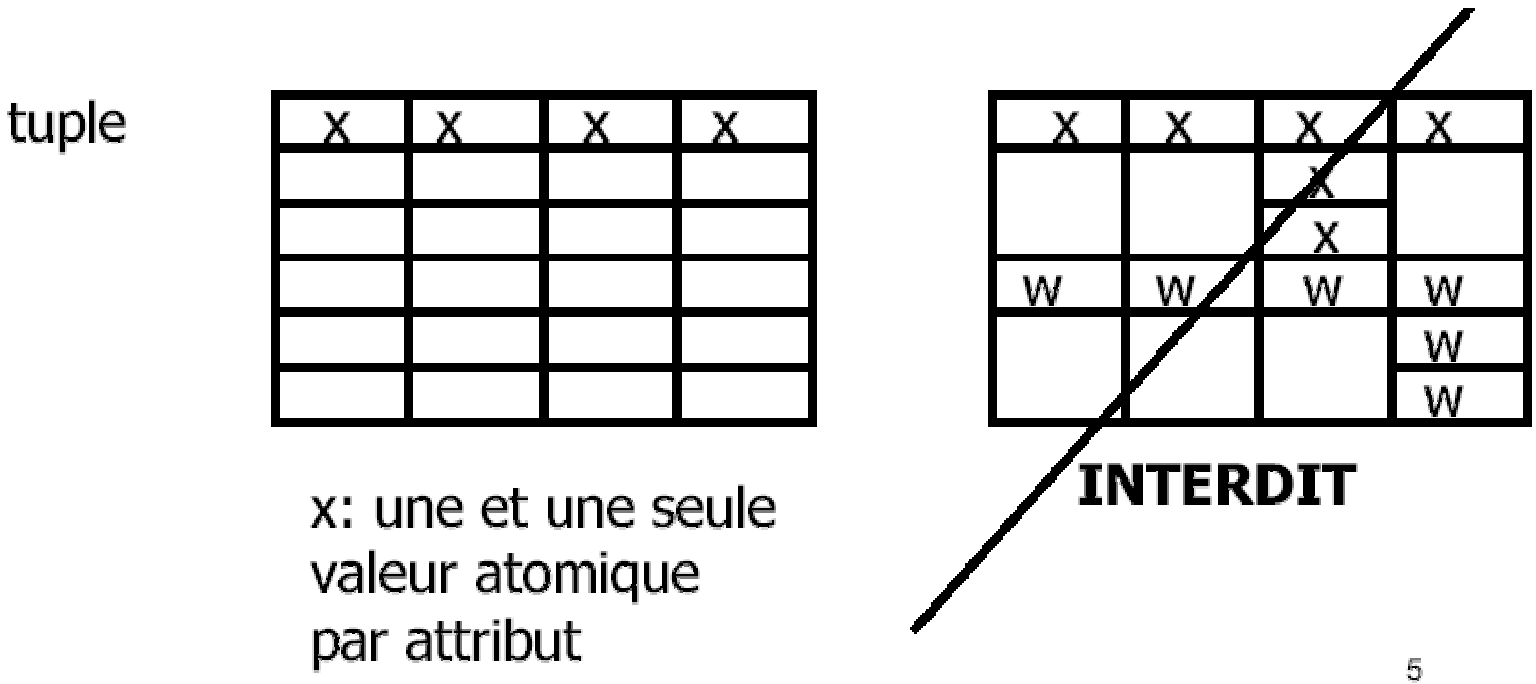
$$Ri = (A1/d1, A2/d2, \ldots, Ay/dy)$$

ou, plus simplement,

$$Ri = (A1, A2, \ldots, Ay)$$

Règles de structuration

- attributs: simples et monovalués (domaine de valeurs atomiques)
- structure plate régulière



Valeurs nulles

- Un attribut peut ne pas être valué pour un tuple: on dit alors qu'il a une valeur nulle
  - exemple: on ne connaît ni l'age d'Annie ni le prénom de Duval

136	Dupont	Jean	19
253	Aubry	Annie	NIL
101	Duval	NIL	21
147	Dupont	Marc	21

Règles d'identification

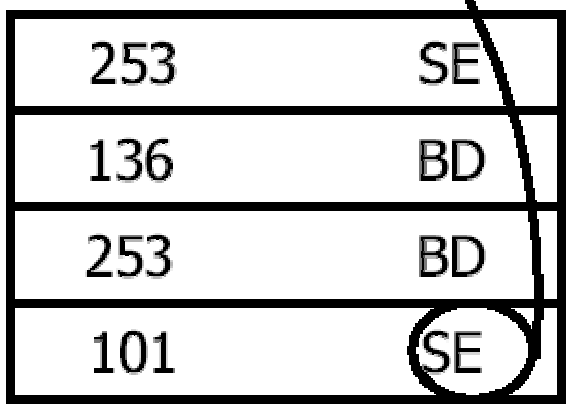
- Toute relation possède un identifiant (clé)
  - il ne peut y avoir deux tuples identiques dans la même relation
- L'identifiant n'admet pas de valeurs nulles

Etudiant	N°Etud	nom	prénom	age
	136	Dupont	Jean	19
	253	Aubry	Annie	20
	101	Duval	André	21
	147	Dupont	Marc	21

Identifiant externe

- Cours (NomC, horaire, prof)

- Suit (N°Etud, NomC)



Suit traduit un TA entre Etudiant et Cours. Elle comporte les identifiants de Etudiant et de Cours. Suit.NomC est un identifiant externe sur Cours.

Domaines de valeurs

- Un domaine est un ensemble de valeurs atomiques que peut prendre un attribut (cf. EA)
- Exemples de domaines:
  - Dnom : chaînes de caractères de longueur maximale 30
  - Dnum : entiers compris entre 0 et 99999
  - Dcouleur : {"bleu", "vert", "jaune"}
  - Dâge : entiers compris entre 16 et 65

Définition d'une relation

- Une relation est définie par :
  - son nom
  - sa liste de couples <nom d'attribut : domaine>
  - son (ses) identifiant(s)
  - la définition de sa sémantique (phrase en français)
- Exemple :

  Etudiant (N°Etud : Dnum, Nom : Dnom,

  Prénom : Dnom, Age : Dâge)

  Identifiant: N°Etud

  Définition: tout étudiant actuellement immatriculé à l'Université

Contraintes de modélisation

- Les notions d'attribut multivalué ou complexe n'existent pas dans le modèle relationnel. Il faut donc les modéliser autrement.
- Pour un attribut complexe, il faut choisir entre le composé ou les composants
- Pour un attribut multivalué, il faut créer une autre relation (ceci pour chaque attribut multivalué)

Représentation d'attribut complexe

- Soit *Adresse : nom rue , n° , ville, NPA*
- Solution 1:
  - un attribut par composant:

    nom rue , n° , ville, NPA

    "Rue de Bourg", "2", "Lausanne", "1003"

  - il est éventuellement possible de définir par ailleurs une vue restituant la notion globale d'adresse
- Solution 2:
  - un attribut Adresse, domaine: chaine de caractères

    "Rue de Bourg 2 Lausanne 1003"

Représentation d'attribut multivalué

Exemple: mémoriser les différents prénoms des étudiants

- INCORRECTE:

  Plusieurs attributs : Prénom1, Prénom2,…

- CORRECTE: créer une relation supplémentaire:

  EtudPrénoms (Num_Etud, Prénom)

136	Jean
136	Marie
101	André
253	Annie
253	Claudine

Ou liste ordonnée:

EtudPrénoms2 (N°Etud, N°Prénom, Prénom)

Identifiant d'une relation

- Une relation peut avoir plusieurs identifiants

EtudPrénoms2 (N°Etud, N°Prénom, Prénom)

- Définition: L'identifiant d'une relation est un ensemble minimum d'attributs de la relation, tel qu'il n'existe pas 2 tuples ayant même valeur pour cet identifiant.
- Règle: tous les attributs de tout identifiant doivent toujours avoir une valeur connue (non nulle).

Identifiants externes

- Décrivent des liens entre relations
- Suit (N°Etud : Dnum, NomC : Dnom)

  N°Etud **référence un** Etudiant

  NomC **référence un** Cours
- Si la relation référencée possède plusieurs identifiants, il faut préciser:

  N°Etud **référence un** Etudiant.N°Etud
- Vérification de l'intégrité référentielle assurée par le SGBD: les identifiants externes désignent nécessairement des tuples existants.

Récapitulatif

- Un schéma relationnel se compose:
  - pour chaque relation de:
    - nom de la relation
    - définition
    - attributs + domaines
    - identifiant(s)
    - éventuellement identifiant(s) externe(s)
    - contraintes d'intégrité associées
  - et des autres contraintes d'intégrité qui portent sur plusieurs relations.

Exemple de schéma relationnel (FormaPerm)

- Domaines:
  - Dnom : chaînes de caractères de longueur inférieure à 30
  - Dch100 : chaînes de caractères de longueur inférieure à 100
  - Dannée : [1970 : 1990]
  - Dnote : [0.0 : 20.0]
  - Ddate : [1 :31] / [1 :12] / [1920 :1990]

- **Relation** : *Personne*

  Attributs: *n°P* : entier sans nul

  nom : *Dnom* sans nul

  adr : *Dch100* sans nul

  Identifiant: (*n°P*)

  Définition: *tout étudiant et tout enseignant de l'école (état actuel).*

- **Relation** : *PersonnePrénoms*

  Attributs: *n°P* : entier sans nul

  prénom : *Dnom* sans nul

  Identifiant: (*n°P* + *prénom*)

  Identifiant externe: *n°P* référence une *Personne*

  Définition: *prénoms des personnes*

- **Relation** : *Etudiant*

  Attributs: *n°P* : entier sans nul

  n°E : entier sans nul

  dateN : *Ddate* sans nul

  Identifiants: (*n°E*) (*n°P*)

  Identifiant externe : *n°P* référence une *Personne*

  Définition: *tout individu qui est actuellement inscrit à l'école, ou qui a déjà passé avec succès un des cours de l'école*

- **Relation** : *EtudiantEtudes*

  Attributs: *n°E* : entier sans nul

  année : *Dannée* sans nul

  diplôme : *Dnom* sans nul

  Identifiant: (*n°E* + *diplôme*)

  Identifiant externe : *n°E* référence un *Etudiant.n°E*

  Définition: *études antérieures des étudiants* car possibilité d 'avoir plusieurs diplômes la même année

- **Relation** : *Enseignant*

  Attributs: *n°P* : entier sans nul

  tel: : entier sans nul

  statut : *Dnom* sans nul

  banque : *Dnom* sans nul

  agence : *Dnom* sans nul

  compte : entier sans nul

  Identifiant: (*n°P*)

  Identifiant externe : *n°P* référence une *Personne*

  Définition: *tout individu assurant actuellement un ou plusieurs cours à l'école*

- **Relation** : *Cours*

  Attributs: *nomC* : *Dnom* sans nul

  cycle : entier sans nul

  n°Ens : entier sans nul

  Identifiant: (*nomC*)

  Identifiant externe : *n°Ens* référence un *Enseignant*

  Définition: *tout cours actuellement offert par l'école*

- **Relation** : *Obtenu*

  Attributs: *n°E* : entier sans nul

  nomC : *Dnom* sans nul

  note : *Dnote* sans nul

  année : *Dannée* sans nul

  Identifiant: (*n°E* + *nomC*)

  Identifiants externes : *n°E* référence un *Etudiant.n°E nomC* référence un *Cours*

  Définition: *l'étudiant n°E a réussi le cours nomC telle année et a obtenu telle note*

- **Relation** : *Inscrit*

  Attributs: *n°E* : entier sans nul

  nomC : *Dnom* sans nul

  Identifiant: (*n°E* + *nomC*)

  Identifiants externes : *n°E* référence un *Etudiant.n°E nomC* référence un *Cours*

  Définition: *actuellement, l'étudiant n°E est inscrit au cours nomC*

- **Relation** : *Prérequis*

  Attributs: *nomC* : *Dnom* sans nul

  nomCprérequis : *Dnom* sans nul

  Identifiant: (*nomC* + *nomCprérequis*)

  Identifiants externes : *nomC* référence un *Cours*
  nomCprérequis référence un *Cours*

  Définition: *le cours nomCprérequis est un prérequis pour le cours nomC*

  Contrainte d'intégrité: *dans tout tuple, nomCprérequis doit être différent de nomC*

Contraintes d'intégrité

- Pour tout tuple de Prérequis <nomC, nomCprérequis>, le cycle de nomCprérequis dans Cours doit être inférieur ou égal à celui de nomC:
  - <x,y> in Prérequis ⇨ x.cycle > y.cycle

- Pour tout tuple de EtudiantEtudes <n°E , année, diplôme>, soit dateN la date de naissance des étudiants dans la relation Etudiant, alors: dateN < année:
  - <x,y,z> in EtudiantEtudes, <a,x,d> in Etudiant

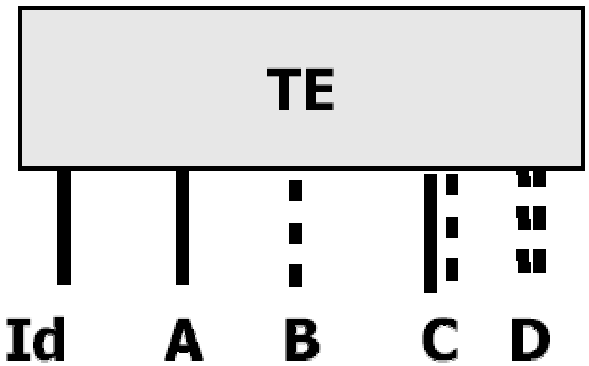
    ⇨ d < y

Contraintes d'intégrité

- Pour tout tuple de Etudiant: cette-année - dateN = 18, où cette-année est une variable SGBD
- Pour tout tuple de Obtenu <n°E, nomC, note, année>, soit dateN la date de naissance de l'étudiant dans la relation Etudiant , alors: dateN < année
- Pour tout tuple de Inscrit <n°E, nomC>, le n°E doit exister dans Obtenu associé à tous les cours existant dans Prérequis associés à nomC.

Transformation d'un schéma conceptuel en schéma logique relationnel

Transformation d'un TE



En relationnel :

TE1 (Id, A, B) A NOT NULL

TE2 (Id, C) C NOT NULL

TE3 (Id, D) D NOT NULL

π [Id] TE2 = π [Id] TE1

π [Id] TE3 $\subseteq$ π [Id] TE1

Attributs simples

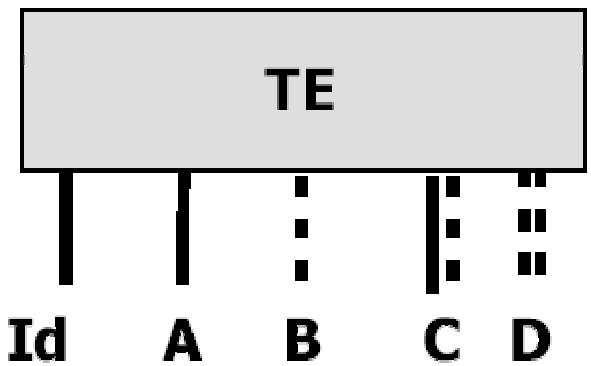
- tous les attributs monovalués peuvent être regroupés dans une même relation

TE1 (Id, A, B) A NOT NULL

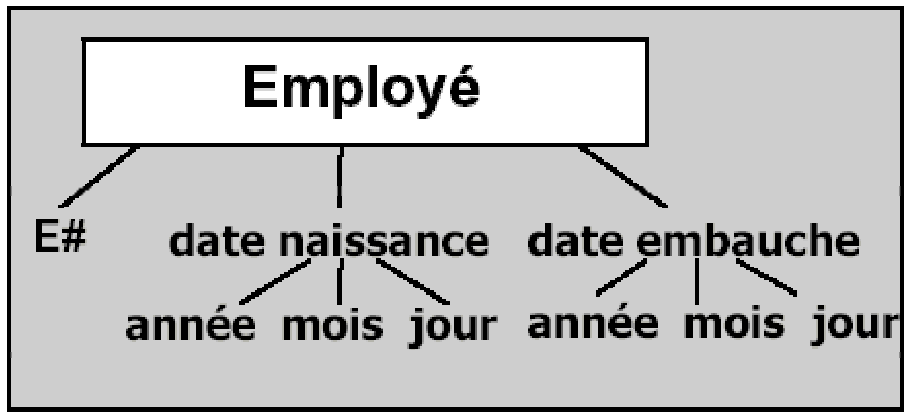
- chaque attribut multivalué demande une relation supplémentaire

TE2 (Id, C) C NOT NULL

TE3 (Id, D) D NOT NULL



Exemple: attributs complexes



Employé

(E#, année-n, mois-n, jour-n, année-e, mois-e, jour-e)

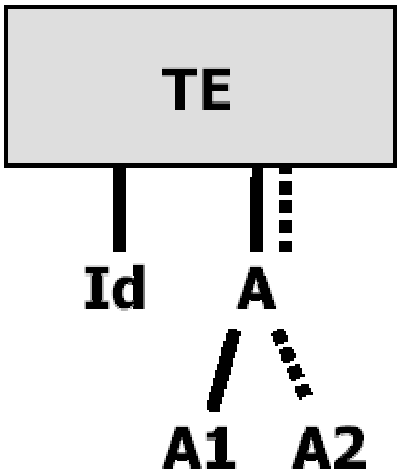
156 1954 12 25 1985 06 10

Employé' (E#, date naissance, date embauche)

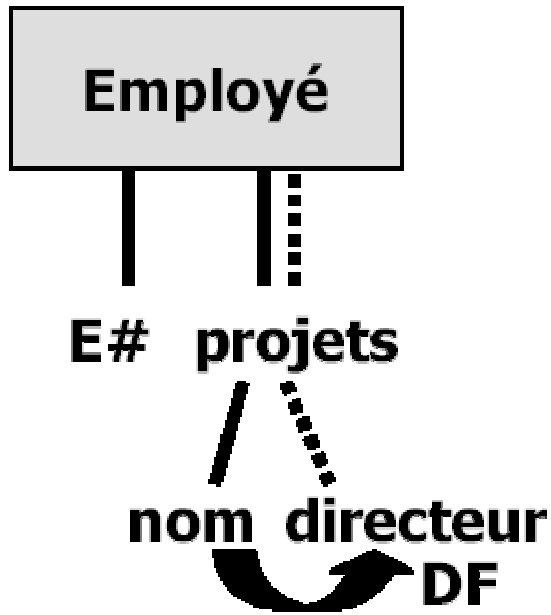
156 19541225 19850610

Attributs complexes multivalués (1)

1) si A1 → A2 : TE1' (Id, A1) & TE2 (A1, A2)



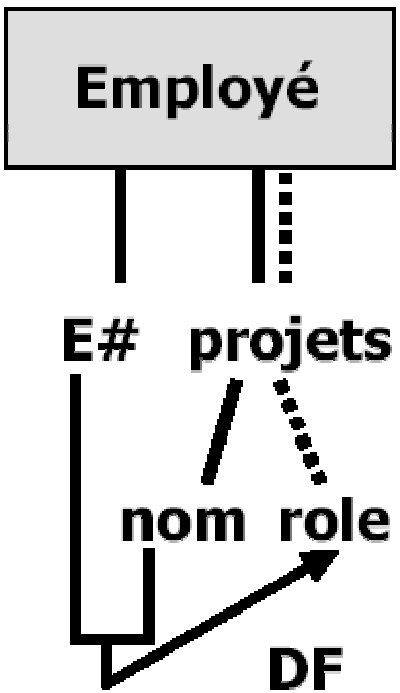
TE (Id, A1, A2)

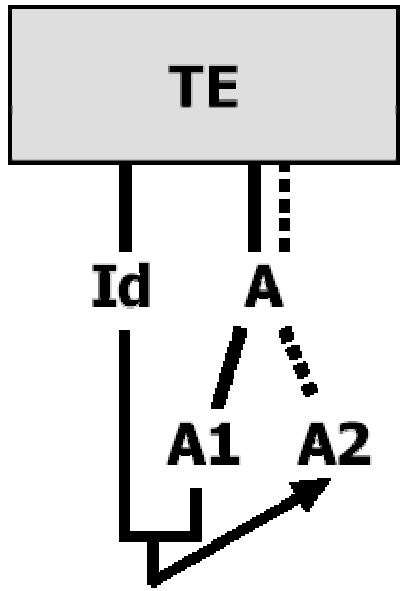


Employé (E#, nomprojet)
Projet (nom, directeur)

Attributs complexes multivalués (2)

2) si (Id, A1) → A2 : TE (Id, A1, A2)





Employé (E#, nomprojet, role)

Contraintes d'intégrité

Personne (Pin, ..., age)
Voiture (V# , ...)
Conduit (Pin, V# , ...)

seules les personnes de plus de 18 ans peuvent conduire des voitures

x ∈ Personne AND (x, v) ∈ Conduit ⇒ x.age > 18

```
SELECT *
FROM   Personne, Conduit
WHERE  Personne.Pin = Conduit.Pin
   AND Personne.age < 18
```

résultat = 0 ⇔

la contrainte d'intégrité est vérifiée

x ∈ Personne AND (x, v) ∈ Conduit

AND x.age < 18 = FALSE

Pour vérifier les CI

- Solution 1:
  - démarrer une transaction
  - exécuter la mise à jour
  - exécuter la requête de validation de la C
  - si résultat ≠ 0 abort de la transaction sinon commit
- Solution 2:
- définir une vue qui respecte la CI
- valider la mise à jour par rapport à la définition de la vue
- s'il y a incohérence abandonner la mise à jour sinon exécuter la mise à jour

Solution 3:

- utiliser les triggers ON UPDATE DO ...

Chapitre 5

Le Langage SQL

L'algèbre relationnelle

Langages de Manipulation

- Langages **formels** : base théorique solide
- Langages **utilisateurs** : version plus ergonomique

- Langages **procéduraux** : définissent comment dériver le résultat souhaité
- Langages **assertionnels** ou déclaratifs : définissent le résultat souhaité

LMD Classiques

- Langages **formels** :
  - Langages algébriques : définissent un ensembled'opérateurs de manipulation
  - Langages prédicatifs (calcul) : définissent le résultat souhaité en utilisant des expressions logiques

- Langages **utilisateurs** BDR :
  - Inspiré des langages algébriques : SQL
  - Inspirés des langages prédicatifs : QBE, QUEL

L'approche algébrique

- Une algèbre est un ensemble d'opérateurs de base, formellement définis, qui peuvent être combinés à souhait pour construire des expressions algébriques
- Une algèbre est dire fermée si le résultat de tout opérateur est du même type que les opérandes (ce qui est indispensable pour construire des expressions)
- Complétude : toute manipulation pouvant être souhaitée par les utilisateurs devrait pouvoir être exprimable par une expression algébrique.

L'algèbre relationnelle

- Opérandes : relations du modèle relationnel
- Fermeture : le résultat de toute opération est une nouvelle relation
- Complétude : permet toutes les opérations nécessaires
- Opérations Unaires (une seule opérande) : sélection (notée σ), projection (notée π), renommage (noté α)
- Opérations binaires : produit cartésien ($\times$), jointures ($\triangleright\triangleleft$),

union ($\cup$), intersection ($\cap$), différence (-), division (/)

Sélection σ

- But : ne retenir que certains tuples de la relation

Pays	nom	capitale	population	surface
	Autriche	Vienne	8	83
	UK	Londres	56	244
	Suisse	Berne	7	41

On ne veut que les pays dont la valeur de surface est inférieure à 100:

Petit-pays = σ [surface < 100] Pays

Petit-pays	nom	capitale	population	surface
	Autriche	Vienne	8	83
	~~UK~~	~~Londres~~	~~56~~	~~244~~
	Suisse	Berne	7	41

Sélection σ

- Opération unaire
- Syntaxe : $\sigma[p]R$
  - P prédicat de sélection (condition de sélection)

<prédicat-élémentaire opérateur logique prédicat-élémentaire>

Opérateur logique $\in \{et, ou\}$

Prédicat élémentaire :

[nom] attribut opérateur de comparaison constante-ou-attribut

Attribut est un attribut de la relation R

Opérateur de comparaison $\in \{\leq, <, \neq, =, >\geq\}$

Sélection σ

- Sémantique : crée une nouvelle relation de population l'ensemble de tuples de R qui satisfont le prédicat p
- Schéma (résultat) = Schéma (opérande)
- Population (résultat) $\subseteq$ population (opérande)

- Exemple : **Petit-pays = σ [surface < 100] Pays**

Petit-pays

nom	capitale	population	surface
Autriche	Vienne	8	83
Suisse	Berne	7	41

Projection π

- But : ne retenir que certains attributs de la relation

Pays	nom	capitale	population	surface
	Autriche	Vienne	8	83
	UK	Londres	56	244
	Suisse	Berne	7	41

On ne veut que les attributs nom et capitale:

Capitales = π [nom, capitale] Pays

Capitales	nom	capitale	population	surface
	Autriche	Vienne	8	83
	UK	Londres	56	244
	Suisse	Berne	7	41

Projection π

- Opération unaire
- Syntaxe : $\pi\left[attributs\right]R$
  - Attributs : liste de l'ensemble d'attributs de R à conserver dans le résultat
- Sémantique : crée une nouvelle relation de population l'ensembles des tuples de R réduits aux seuls attributs de la liste spécifiée
- Schéma (résultat)$\subseteq$ Schéma (opérande)
- Nb tuples (résultat) $\leq$ nb tuples (opérande)

Sélection/Projection

- On veut les capitales des petits pays :
  - **Petit-pays = σ [surface < 100] Pays**
  - **Capitales = π [nom, capitale] Pays**

Capitale-petit-pays =

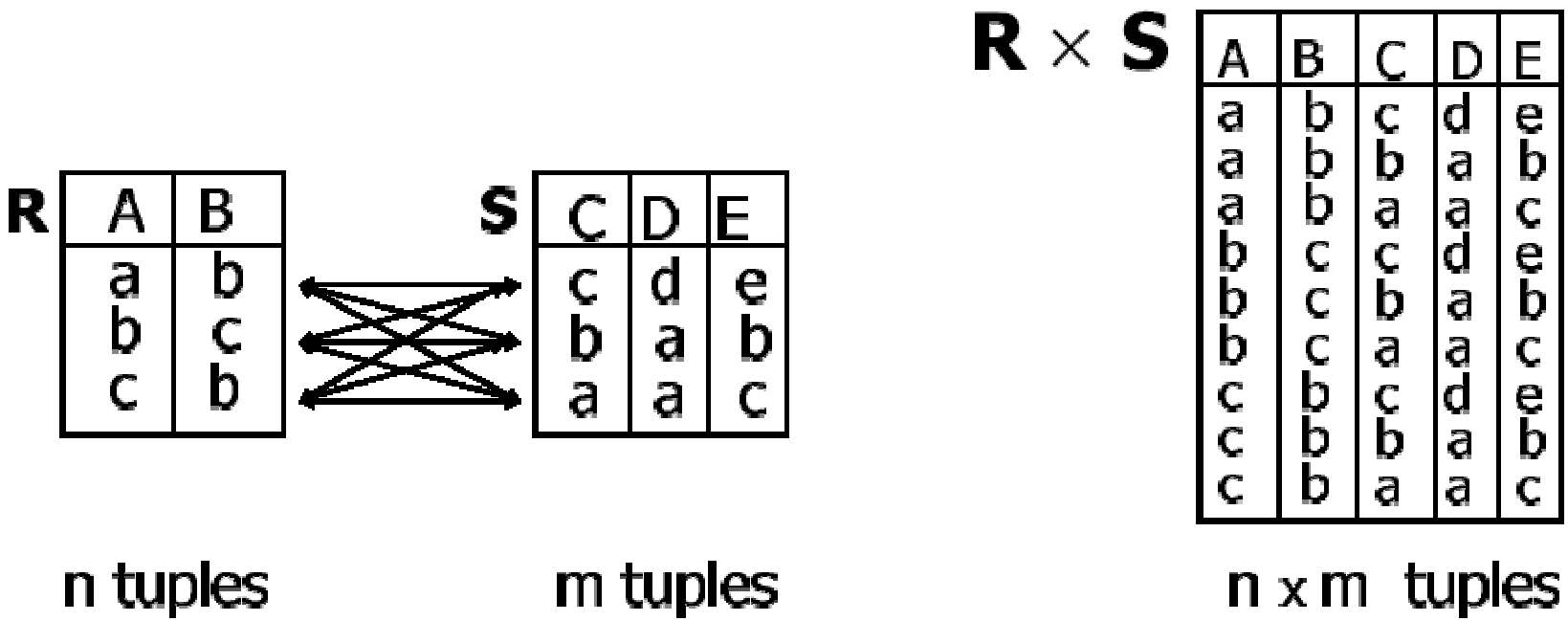
π [nom, capitale] σ [surface < 100] Pays

nom	capitale	population	surface
Irlande	Dublin	3	70
Autriche	Vienne	8	83
UK	Londres	56	244
Suisse	Berne	7	41

(Parties grise et beige à enlever)

Produit Cartésien $\times$

- But : construire toutes les combinaisons de tuples de deux relations (en général en vue d'une sélection)
- Syntaxe : $R \times S$
- Exemple :

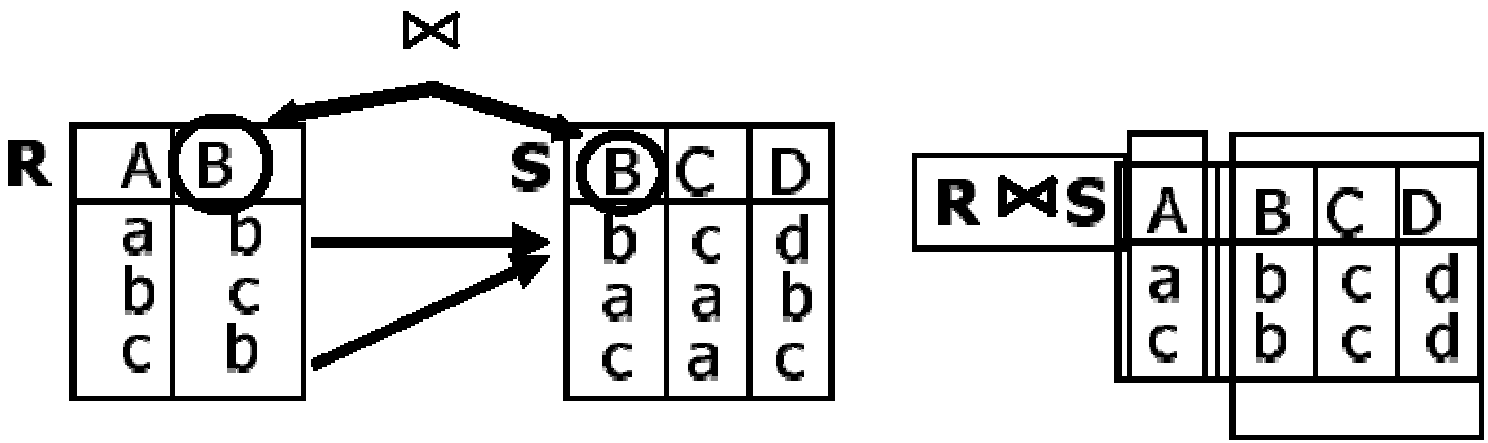


Produit Cartésien $\times$

- Opération binaire
- Sémantique : chaque tuple de R est combiné avec chaque tuple de S
- Schéma : Schéma (R $\times$ S)=Schéma(R) $\cup$ Schéma(S)
- Précondition : R et S n'ont pas d'attributs de même nom (sinon renommage des attributs avant de faire le produit)

Jointure Naturelle ▷◁

- But : créer toutes les combinaisons significatives entre tuples de deux relation
  - Significatif : portent la même valeur pour les attributs de même nom
- Précondition : les deux relations ont au moins un attribut de même nom
- Exemple

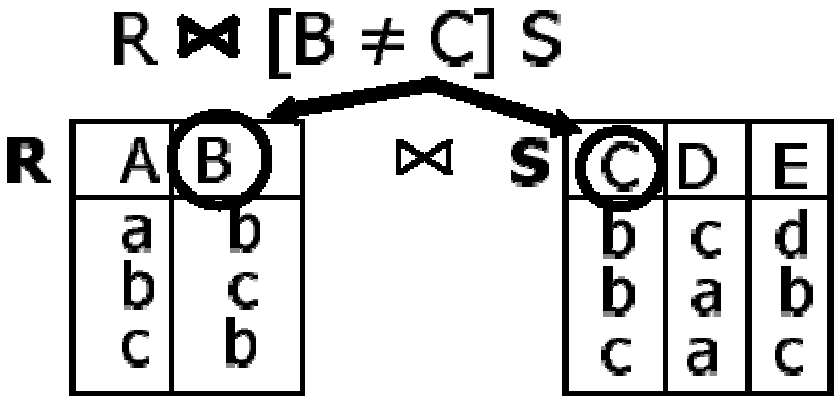


Jointure Naturelle $\triangleright\triangleleft$

- Opération binaire
- Syntaxe : $R \triangleright\triangleleft S$
- Sémantique : combine certains tuples
- Schéma : Schéma (R $\triangleright\triangleleft$ S)=Schéma(R)$\cup$ Schéma(S)
  - Les attributs de même nom, n'apparaissent qu'une seule fois
- La combinaison exige l'égalité des valeurs de tous les attributs de même nom de R et de S
  - Si R et S n'ont pas d'attributs de même nom la jointure peut être dynamiquement remplacée par un produit cartésien

Theta-Jointure $\triangleright\triangleleft [p]$

- But : créer toutes les combinaisons significatives entre tuples de deux relation
  - Significatif : critère de combinaison explicitement défini en paramètre de l'opération
- Précondition : les deux relations n'ont pas d'attribut de même nom
- Exemple



R⋈ [B ≠ C] S

A	B	C	D	E
a	b	c	a	c
b	c	b	c	d
b	c	b	a	b
c	b	c	a	c

Theta-Jointure $\triangleright\!\triangleleft\left[p\right]$

- Opération binaire
- Syntaxe : $R \triangleright\!\triangleleft \left[p\right] S$
  - p: prédicat/condition de jointure
  - <prédicat-élémentaire et/ou prédicat élémentaire>
- Sémantique : combine les tuples qui satisfont le prédicat
- Schéma : Schéma $R \triangleright\!\triangleleft \left[p\right] S$ =Schéma(R) $\cup$ Schéma(S)

Union $\cup$

- Opération binaire
- Syntaxe : $R \cup S$
- Sémantique : réunit dans une même relation les tuples de R et ceux de S
- Schéma : Schéma $R \cup S$ =Schéma(R)=Schéma(S)
- Précondition : Schéma(R)=Schéma(S)
- Exemple :

R1

A	B
a	b
b	b
y	z

R2

A	B
u	v
y	z

R1 $\cup$ R2

A	B
a	b
b	b
y	z
u	v

Intersection $\cap$

- Opération binaire
- Syntaxe : $R \cap S$
- Sémantique : sélectionne les tuples qui sont à la fois dans R et S
- Schéma : Schéma $R \cap S$ =Schéma(R)=Schéma(S)
- Précondition : Schéma(R)=Schéma(S)
- Exemple :

R1

A	B
a	b
y	z
b	b

R2

A	B
u	v
y	z

R1 $\cap$ R2

A	B
y	z

Différence —

- Opération binaire
- Syntaxe : $R - S$
- Sémantique : sélectionne les tuples de R qui ne sont pas dans S
- Schéma : Schéma $R - S$ =Schéma(R)=Schéma(S)
- Précondition : Schéma(R)=Schéma(S)
- Exemple :

R1

A	B
a	b
y	z
b	b

R2

A	B
u	v
y	z

R1 – R2

A	B
a	b
b	b

Renommage α

- But : résoudre des problèmesde compatibilité entre noms d'attributs de deux relations opérandes d'une opération binaire
- Opération unaire
- Syntaxe : $\alpha\left[nom\ attribut : nouveau\ nom\right]R$
- Sémantique : les tuples de R qui ne sont pas dans S
- Schéma : Schéma $\alpha\left[n,m\right]R$ =Schéma(R)
- Précondition : le nouveau nom n'existe pas déjà dans R
- Exemple : $R2 = \alpha\left[B,C\right]R1$

R1

A	B
a	b
y	z
b	b

R2

A	C
a	b
y	z
b	b

Division /

But : traiter les requêtes du style « les … tels que TOUS les… »

- Soient R(A1, …An) et V(A1, …, Am) avec n>m et A1, …, Am des attributs de même nom dans R et V
- $$R / V = \left\{ \begin{array}{l} < am+1, am+2, ..., an > / \forall < a1, a2, ..., am > \in V, \\ \exists < a1, a2, ..., am, am+1, am+2, ..., an > \in R \end{array} \right\}$$

- Exemple :

R

A	B	C
1	1	1
1	2	0
1	2	1
1	3	0
2	1	1
2	3	3
3	1	1
3	2	0
3	2	1

V

B	C
1	1
2	0

R/V

A
1
3

V'

B	C
1	1

R/V'

A
1
2
3

V''

B	C
3	5

R/V''

A
/

SQL :

Structured Query Langage

"Le" langage des bases de données relationnelles

SQL: une norme

- SQL-2
- SQL-3 en cours
- Norme: langage complexe
  - 150 pages
- Langage non orthogonal
  - plusieurs façons d'exprimer la même requête
- Un seul paradigme pour des fonctionnalités diverses

SQL : trois langages

- Langage de requêtes
- SELECT ....
- Langage de manipulation de données
  - insertions de tuples: INSERT
  - mises à jour des tuples: UPDATE
  - suppressions de tuples: DELETE
- Langage de définition de données (schéma)
  - création de relations : CREATE TABLE
  - modification du schéma: ALTER TABLE
  - suppression de relations: DROP TABLE
  - vues, index .... : CREATE VIEW ...

CREATE TABLE

- Commande créant une relation
- CREATE TABLE nom_relation

  (nom_col type_col [DEFAULT valeur] [col_contrainte])*

  _ table_contrainte) | AS subquery ;

Legende :

a |b : a ou b, [option];

* : applicable autant de fois que souhaité; mots en capitale : mots-clé.

CREATE TABLE

PAYS			
nom	capitale	population	surface
Ireland	Dublin	3	70
Austria	Vienna	8	83
United Kingdom	London	56	244
Switzerland	Berne	7	41

CREATE TABLE PAYS

(nom VARCHAR(20) NOT NULL,

capitale VARCHAR(15),

population DECIMAL(4,2),

surface DECIMAL(5,0),

....)

Contraintes

- Table_contrainte :
  - UNIQUE (nom_col)*
  - PRIMARY KEY (nom_col)* (incompatible avec UNIQUE)
  - FOREIGN KEY (nom_col)* REFERENCES nom_table [(nom_col)*]
- Col-contrainte :
  - [NOT] NULL
  - UNIQUE
  - PRIMARY KEY (incompatible avec UNIQUE)

CREATE VIEW

CREATE VIEW olympic_capital

(when, place, country, capital)

AS

SELECT year, location, olympics.country, capital

FROM Olympics, Pays

WHERE Olympics.country = Pays.nom

Pays

nom	capitale	population	surface
Ireland	Dublin	3	70
Austria	Vienna	8	83
United Kingdom	London	56	244
Switzerland	Berne	7	41
U.S.A.	Washington	987	852

Olympics

year	location	country
1896	Athens	Greece
1900	Paris	France
1904	St.Louis	U.S.A.
1908	London	United Kingdom

⇒ select *

from olympic_capital

Olympic capital

when	place	country	capital
1904	St.Louis	U.S.A.	**Washington**
1908	London	**United Kingdom**	London

Create view

- Définir une vue basée sur une ou plusieurs tables ou vues.
- CREATE [OR REPLACE] VIEW nom_vue

[(nv_nom_col)*]

AS subquery [WITH READ ONLY];

DROP et RENAME

- DROP : Supprimer une relation ou une vue;
- DROP TABLE nom_table;
- DROP VIEW nom_vue;
- RENAME : renommer une relation ou une vue;
- RENAME {nom_table _ nom_vue } TO nv_nom;

ALTER TABLE

- Modifier la définition d'une relation :
  - Ajouter une colonne ou une contrainte,
  - mot clé : ADD,
  - Modifier une colonne ou une contrainte,
  - mot clé : MODIFY,
  - Renommer une relation,
  - mot clé : RENAME.

ALTER TABLE

ALTER TABLE nom_table

{ RENAME TO nv_nom_table _

ADD (

[nom_col type_col [DEFAULT valeur] [col_contrainte]),]*

[table_contrainte]) _

MODIFY

(nom_col [type_col] [DEFAULT valeur] [col_contrainte])*

}

ALTER VIEW

- Recalculer une vue.
- ALTER VIEW nom_vue COMPILE;

Manipulation des données

- INSERT INTO: ajouter un tuple dans une relation ou vue
- UPDATE: changer les tuples d'une relation ou vue
- DELETE FROM: éliminer les tuples d'une relation ou vue.

INSERT INTO

INSERT INTO olympics

VALUES (1996, ‘Atlanta’, ‘U.S.A’)

Olympics		
year	**location**	**country**
1896	Athens	Greece
1900	Paris	France
1904	St.Louis	U.S.A.
1908	London	U.K.
1996	*Atlanta*	*U.S.A.*

INSERT INTO olympics

(year, location)

VALUES (1996, ‘Atlanta’)

Olympics		
year	**location**	**country**
1896	Athens	Greece
1900	Paris	France
1904	St.Louis	U.S.A.
1908	London	U.K.
1996	*Atlanta*	*null*

INSERT

INSERT INTO {nom_table _ nom_vue}

[(nom_col)*]

{ VALUES (valeur)* _ sous-requête };

UPDATE

UPDATE country

SET capital = ‘Londres’

WHERE country = ‘Ireland’

UPDATE country

SET drive = ‘L’, rainfall = rainfall/2

WHERE country <> ‘France’ AND drive = ‘R’

Country

country	capital	population	area
Ireland	Dublin	3	70
Austria	Vienna	8	83
Utd Kingdom	London	56	244
Switzerland	Berne	7	41

UPDATE

- UPDATE {nom_table _ nom_vue} SET

{ (nom_col)* = (sous-requête) _

nom_col = { valeur _ (sous-requête)} }*

WHERE condition;

DELETE FROM

DELETE FROM country

WHERE population > 50

⇨éliminer les pays trop peuplés

Country

country	capital	population	area
Ireland	Dublin	3	70
Ausirla	Vienna	8	83
Utd Kingdom	London	58	244
Switzerland	Berne	7	41

DELETE FROM country

⇨ la fin du Monde

DELETE

•DELETE FROM {nom_table _ nom_vue}

WHERE condition;

SQL : partie langage de requêtes

Structure de base d.une requête : le BLOC

SELECT A1, …, An

FROM R1, …, Rm

WHERE [condition]

avec Ai un nom d'attribut

Rj un nom de relation

Exemple de requête

SELECT nom, capitale, population

FROM Pays

WHERE population < 20 ;

Pays

nom	capitale	population	surface
Ireland	Dublin	3	70
Austria	Vienna	8	83
UK	London	56	244
Switzerland	Berne	7	41

(Partie grisée à enlever)

Exemple de requête

SELECT * FROM country

⇨ tous les attributs de tous les tuples dans la relation “country”

Country			
country	capital	population	area
Ireland	Dublin	3	70
Austria	Vienna	8	83
Utd Kingdom	London	56	244
Switzerland	Berne	7	41

Algèbre ⇨ SQL : Sélection

En algèbre :

$\sigma_C(R)$

En SQL :

```
SELECT *
FROM R
WHERE C
```

Algèbre ⇨ SQL : Projection

En algèbre :

$\pi_{A1,A2,..An}(R)$

En SQL :

```
SELECT A1,A2,..An
FROM R
```

Algèbre ⇨ SQL: Sélection+Projection

H := π [country, capital, population] (σ [population < 20] Country)

Country

country	capital	population	area
Ireland	Dublin	3	70
Austria	Vienna	8	83
Utd Kingdom	London	56	244
Switzerland	Berne	7	41

SELECT country, capital, population

FROM country

WHERE population < 20

Algèbre ⇨ SQL : Renommage

En algèbre :

δA1/B1, ..., An/Bn(R)

En SQL :

Impossible de renommer des attributs. Il faut faire des « *copies logiques* » des relations.

SELECT *

FROM R, **R R2**

WHERE R.A = R2.A ..

R2 peut être vue comme une copie logique de R

Algèbre ⇨ SQL : Produit cartésien

En algèbre :

$R \times S$

En SQL :

```
SELECT *
FROM R, S
```

Algèbre ⇨ SQL : Jointure

En algèbre :

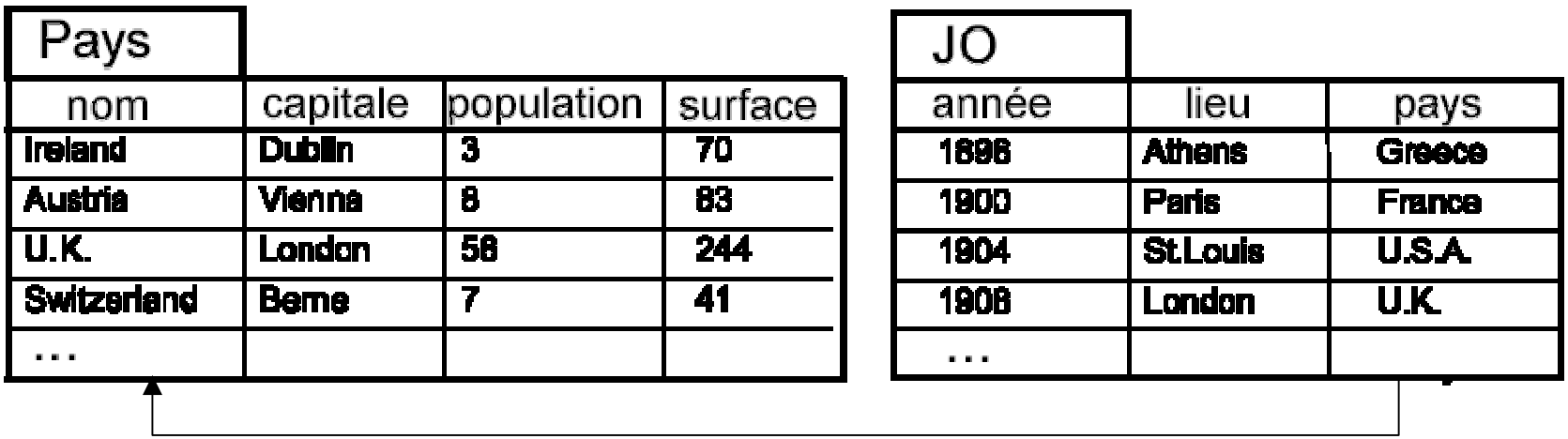
$R \triangleright\!\!\triangleleft S$

En SQL :

```
SELECT *
FROM R, S
WHERE R.A1 = S.A1
AND R.A2 = S.A2
...
AND R.An = S.An
```

Avec A1, ., An tous les attributs communs à R et S

Jointure de 2 relations



Pays

nom	capitale	population	surface
Ireland	Dublin	3	70
Austria	Vienna	8	83
U.K.	London	58	244
Switzerland	Berne	7	41
...			

JO

année	lieu	pays
1896	Athens	Greece
1900	Paris	France
1904	St.Louis	U.S.A.
1908	London	U.K.
...		

SELECT année, lieu, pays, capitale

FROM JO, Pays

WHERE JO.pays = Pays.nom ;

Jointure sur la même relation

SELECT P1.nom, P2.nom, P1.capitale

FROM Pays P1, Pays P2

WHERE P1.capitale = P2.capitale ;

⇨Toutes les paires de pays ayant le même nom de capitale (plus les doubles)

NB: La relation Pays est renommée en P1 et P2

Algèbre ⇨ SQL : Union

En algèbre :

R ∪ S

En SQL :

BlocR *union* BlocS

```
SELECT *
FROM R
UNION
SELECT *
FROM S
```

Algèbre ⇨ SQL : Intersection

En algèbre :

$R \cap S$

En SQL :

BlocR *intersect* BlocS

```
SELECT *
FROM R
INTERSECT
SELECT *
FROM S
```

Algèbre ⇨ SQL : Différence

En algèbre :

R - S

En SQL :

BlocR *minus* BlocS

```
SELECT *
FROM R
MINUS
SELECT *
FROM S
```

Différences

- Entre le modèle relationnel (sa structure) en pratique et en théorie.
- En SQL (faux en algèbre) :
  - une relation peut contenir plusieurs occurrences d'un n-uplet,
  - une relation peut être triée,
  - il existe une valeur spéciale dite **indéfinie** (null) utilisée pour remplir un champ dont on ne connaît pas la valeur.

Remarques

• En SQL, pour n'avoir qu'une seule occurrence de chaque n-uplet dans une relation : **distinct.**

Exemple : select distinct A, B from R

• En SQL, le produit cartésien est possible sans renommer les attributs communs. Ex : schéma(R×S) = A (de R), B (de R), B (de S), C (de S).

• En SQL, si plusieurs attributs ont le même nom, pour résoudre l'ambiguité, on spécifie la relation auquel l'attribut appartient.

Ex : select A, R.B, C from R,S

Opérateurs de conditions

- Opérateurs utilisés pour construire les conditions des blocs.
- Ces opérateurs peuvent être :
  - de comparaison
  - logiques
  - ensemblistes

Opérateurs de comparaison

= égal	WHERE surface **=** 200
<> non égal	WHERE capitale **<>** ‘Paris’
> plus grand que	WHERE population **>** 8
>= plus grand ou égal	WHERE population **>=** 8
< plus petit que	WHERE surface **<** 83
<= plus petit ou égal	WHERE surface **<=** 83

Opérateurs logiques

- Tous les prédicats: **AND**

  WHERE population<10 **AND** surface<500

- Un des prédicats: **OR**

  WHERE population<10 **OR** surface<500

- Négation de la condition : NOT

```
SELECT P1.nom, P2.nom, P1.capitale
FROM Pays P1, Pays P2
WHERE P1.capitale = P2.capitale
AND NOT P1.nom = P2.nom ;
```

NOT (exclusion)

NOT BETWEEN

NOT LIKE

NOT IN

NOT ALL

WHERE country **NOT LIKE** '%land%

⇨ Exclusion de Netherlands, Ireland, Finland, Poland, England, Switzerland ...

IN

- Dans un ensemble :

  WHERE currency = ‘Pound’ **OR**
  currency = ‘Shilling’ **OR**
  currency = ‘Franc’

- Equivalent à:

  WHERE currency **IN (‘Pound’, ‘Shilling’, ‘Franc’)**

BETWEEN

Entre 2 valeurs :

```
WHERE    population >= 50 AND
         population <= 60
```

- Equivalent à:

```
WHERE    population
         BETWEEN 50 AND 60
```

Conditions partielles (wildcards)

% : zéro ou n'importe quel caractère

WHERE country **LIKE '%land'**

⇨ England, Ireland, Iceland

WHERE country **LIKE '%ran%'**

⇨ Iran, France

_ : exactement un caractère

WHERE country **LIKE 'I_eland'**

⇨ Ireland, Iceland

Valeurs calculées

```
SELECT country, population, area, birth_rate
FROM country
WHERE       (population * 1000 / area) < 50
            AND (population * birth_rate / area) > 0
```

Opérateurs: +, -, *, /

La valeur « NULL »

Une valeur inconnue ou non définie:

SELECT country

FROM country

WHERE mountain **IS NULL**

⇨ Netherlands

SELECT country

FROM country

WHERE mountain **IS NOT NULL**

⇨ Austria, Switzerland

Requêtes: prédicats de sélection

SELECT [columns]

FROM [tables]

WHERE [conditions]

WHERE couleur **=** rouge **OR (** poids < 60 **AND** poids > 50**)**

WHERE country **LIKE** '%land'

WHERE couleur **IN (**'bleu'**,** 'rouge'**,** 'blanc'**)**

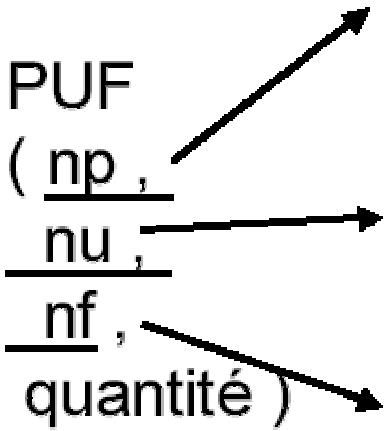
WHERE poids **BETWEEN** 50 **AND** 60

WHERE **NOT** LIKE '%land%

WHERE (population ***** 1000 **/** area) **<** 50

WHERE couleur **IS NULL**

BD Exemple : Livraisons



P (np , nomp , couleur , poids , prix)
les produits

U (nu , nomu , ville , pays)
les usines

F (nf , nomf , type , ville , pays)
les fournisseurs

PUF (np, nu, nf , quantité)
les livraisons

Blocs emboîtés : IN

Nom et couleur des produits livrés par le fournisseur 1

Solution 1 : la jointure

```
SELECT DISTINCT nomp, couleur FROM P, PUF
WHERE PUF.np = P.np AND nf = 1 ;
```

Solution 2 : le bloc emboîté

```
SELECT nomp, couleur FROM P
WHERE np IN
      ( SELECT np
        FROM PUF
        WHERE nf = 1) ;
```

Ensemble des produits livrés par le fournisseur 1

Nom et couleur des produits du fournisseur 1

Composition de conditions

Nom des fournisseurs qui approvisionnent une usine de Londres ou Paris en produit rouge

```
SELECT nomf FROM F
WHERE nf IN
        (SELECT nf
        FROM PUF
        WHERE np IN
                (SELECT np FROM P
                WHERE couleur = 'Rouge')
        AND nu IN
        (SELECT nu FROM U
        WHERE ville = 'Londres OR ville = 'Paris')) ;
```

```
SELECT nomf
  FROM  PUF, P, F, U
  WHERE couleur = 'rouge'
        AND PUF.np = P.np
        AND PUF.nf = F.nf
        AND PUF.nu = U.nu
        AND  (U.ville = 'Londres'
              OR U.ville = 'Paris');
```

Au moins un : ANY (ou SOME)

Ensemble des numéros de fournisseurs de produits rouges

```
SELECT nf
FROM PUF
WHERE np = ANY
        ( SELECT np FROM P
        WHERE couleur = 'Rouge' ) ;
```

Equivalent au **IN**

Numéros des produits moins lourds qu'au moins un produit du fournisseur 1

```
SELECT np
FROM P
WHERE poids < ANY
        ( SELECT poids FROM P, PUF
        WHERE P.np = PUF.np AND nf = 1 ) ;
```

Tous : ALL

Ensemble des fournisseurs de produits rouges uniquement

```
SELECT nf
FROM F
WHERE rouge = ALL
        ( SELECT couleur
        FROM P
        WHERE np IN
                ( SELECT np
                FROM PUF
                WHERE PUF.nf = F.nf) ) ;
```

Conditions sur des ensembles: CONTAINS

- Inclusion des ensembles

Noms des fournisseurs qui fournissent tous les produits rouges

```
SELECT nomf
FROM F
WHERE ( SELECT np
        FROM PUF
        WHERE PUF.nf = F.nf )
    CONTAINS
    ( SELECT np
        FROM P
        WHERE couleur = ' rouge ');
```

Ensemble des produits du fournisseur nf

Ensemble des produits rouges

Condition sur UN ensemble : EXISTS

- Test si l'ensemble n'est pas vide

Noms des fournisseurs qui fournissent au moins un produit rouge

```
SELECT nomf
FROM F
WHERE EXISTS
        ( SELECT *
        FROM PUF, P
        WHERE PUF.nf = F.nf
                AND couleur = ' rouge '
                AND PUF.np = P.np ) ;
```

Traitement des résultats

- Fonctions sur les colonnes
- Elimination de tuples répétés (DISTINCT)
- Définition de l'ordre des tuples (ORDER BY)
- Regroupement de résultats (GROUP BY)

Fonctions sur des colonnes

- Attributs calculés

  Ex: SELECT nom, population*1000/surface FROM Pays

- Opérateurs sur attributs numériques
  - SUM: somme des valeurs des tuples sélectionnés
  - AVG: moyenne
- Opérateurs sur tous types d'attributs
  - MIN: minimum
  - MAX: maximum
  - COUNT: nombre de tuples sélectionnés

Exemple

Pays

nom	capitale	population	surface	Continent
Ireland	**Dublin**	**3**	**70**	Europe
Austria	**Vienna**	**8**	**83**	Europe
U.K.	**London**	**58**	**244**	Europe
Switzerland	**Berne**	**7**	**41**	Europe
Canada	Ottawa	31	9 975	Amérique
...				

SELECT **MIN**(population),

MAX(population),

AVG(population),

SUM(surface),

COUNT(*)

FROM Pays

WHERE continent = 'Europe' ;

⇨un tuple avec la population du plus petit pays d'Europe, la pop. du pays le plus grand, la moyenne des pop. De tous les pays d.Europe, la somme des surfaces et le nombre de pays d'Europe

DISTINCT

Pays

nom	capitale	population	surface	continent
Ireland	**Dublin**	**3**	**70**	Europe
Austria	**Vienna**	**8**	**83**	Europe
U.K.	**London**	**56**	**244**	Europe
Switzerland	**Berne**	**7**	**41**	Europe
Canada	Ottawa	31	9 975	Amérique
...				

Suppression des doubles

SELECT **DISTINCT** continent

FROM Pays ;

ORDER BY

Pays				
nom	capitale	population	surface	continent
Ireland	Dublin	3	70	Europe
Austria	Vienna	8	83	Europe
U.K.	London	56	244	Europe
Switzerland	Berne	7	41	Europe
...				

Tri des tuples de la table résultat

SELECT continent, country, population

FROM country

WHERE area > 60

ORDER BY continent, country **DESC**

GROUP BY

Pays

nom	capitale	population	surface	continent
Ireland	**Dublin**	**3**	**70**	Europe
Austria	**Vienna**	**8**	**83**	Europe
U.K.	**London**	**56**	**244**	Europe
Switzerland	**Berne**	**7**	**41**	Europe
Canada	Ottawa	31	9 975	Amérique
...				

Partition de l'ensemble des tuples en groupes

SELECT continent, MIN(population), MAX(population),

AVG(population), SUM(surface), COUNT(*)

FROM Pays

GROUP BY continent ;

⇒ un tuple pour chaque continent, avec le min, le max, la moyenne de population et la surface sur l'ensemble des pays le constituant

CREATE TRIGGER salary_check

```
BEFORE INSERT OR UPDATE OF sal, job ON employee
FOR EACH ROW
        WHEN (new.job <> 'PRESIDENT')
        DECLARE minsal NUMBER; maxsal NUMBER;
        BEGIN
                SELECT minsalary, maxsalary
                INTO minsal, maxsal
                FROM salary_guide
                WHERE job = new.job
          IF ( new.sal < minsal OR new.sal > maxsal )
          THEN raise_application_error (20601)
          ENDIF;
        END;
```

CREATE TRIGGER nom-du-trigger

BEFORE | AFTER | *INSTEAD OF*

INSERT | UPDATE | DELETE OF noms-d'attributs ON nomtable

FOR EACH ROW | STATEMENT

WHEN (condition)

<bloc PL/SQL ou programme Java ou C >

Comment sont utilisés les « Triggers »

- Éviter les transactions invalides
- Améliorer les sécurités (autorisations
- Renforcer l'intégrité de la base de données dans le cas d'une BD distribuée
- Renforcer les règles complexes
- Fournis des logs d'evènements
- Fournis des outils d'audit
- Maintien les tables dupliquées synchrones
- Fournit des Stats. Sur les accès aux tables

Précaution d'utilisation des « Trigger »

Bien que les triggers soient très utiles pour personnaliser une base de données, il ne faut les utilser que dans la nécessité.

Une utilisation excessive peut engendrer des interdépendances complexes, qui rend alors la base difficile à maintenir.

Par exemple, lorsque qu'un trigger est lancé, il peut faire appel à d'autres triggers. Ceci conduit à une cascade de triggers.

Conclusion

- Langage très puissant

  Il existe de nombreuses autres fonctions

- Requêtes indépendantes de l'implémentation des tables (index, fichiers ...)

- Optimisation des requêtes par le SGBD

- Extensions :

  SQL 3 : relationnel-objet avec spatial, mutimedia, séries temporelles ...

Chapitre 6

Normalisation d'une base de données relationnelle

Qu'est-ce qu'une BD relationnelle « correcte » ?

Un ensemble de relations tel que :

- chaque relation décrit un fait élémentaire avec les seuls attributs qui lui sont directement liés
- il n'y a pas de redondance d'information, génératrices de problèmes lors des MAJ
- il n'y a pas de perte d 'information

Personne (nom, prénom), Adresse (no, rue, ville)

Qui habite oú ? Impossible de répondre !

Qu'est-ce qu'une BD relationnelle « incorrecte » ?

Une relation n'est pas correcte si :

- elle implique des répétitions au niveau de sa population
- elle pose des problèmes lors des MAJ insertions / modifications / suppressions
- Les conditions pour qu'une relation soit correcte peuvent être définies formellement:

⇨règles de normalisation

Exemple de mauvaise relation en 1FN

Livraison (N°fourn,	adr.f,	N°prod,	prix-p,	qté)
3	Lausanne	52	65	10
22	Bienne	10	15	5
22	Bienne	25	10	12
3	Lausanne	25	10	5
3	Lausanne	10	15	20

- L'adresse du fournisseur ne dépend pas du produit.
- Le prix du produit ne dépend pas du fournisseur

  $\Rightarrow$ **REDONDANCES**

- Anomalies de mise à jour

Exemple de mauvaise relation en 1FN

Livraison (N°fourn,	adr.f,	N°prod,	prix-p,	qté)
3	Lausanne	52	65	10
22	Bienne	10	15	5
22	Bienne	25	10	12
3	Lausanne	25	10	5
3	Lausanne	10	15	20

Si un fournisseur change d'adresse et qu'un seul tuple (une seule ligne) est mis à jour:

⇒**incohérence**

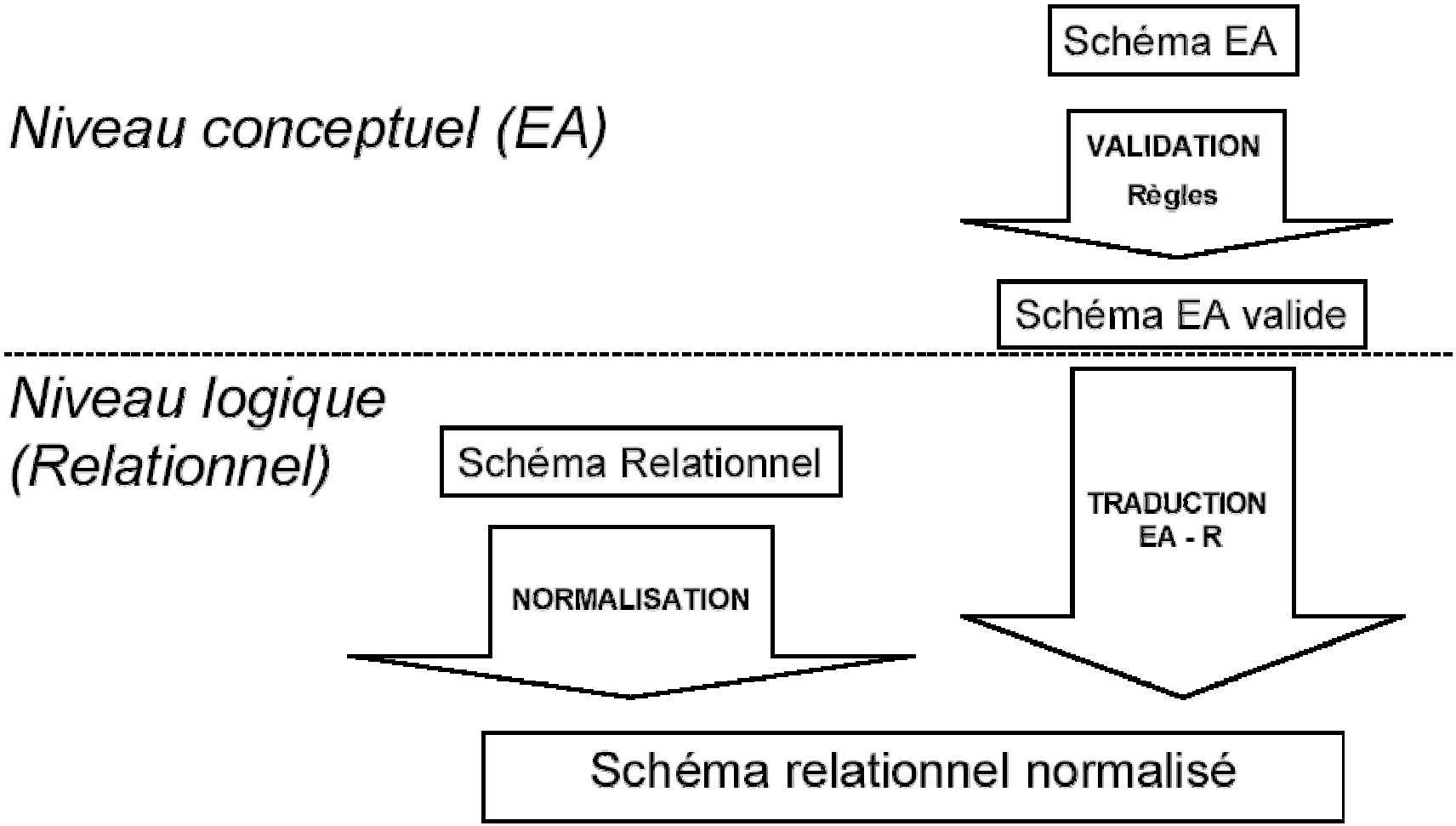
Si une nouvelle ligne est insérée pour un fournisseur connu, mais avec une adresse différente:

⇒ **incohérence**

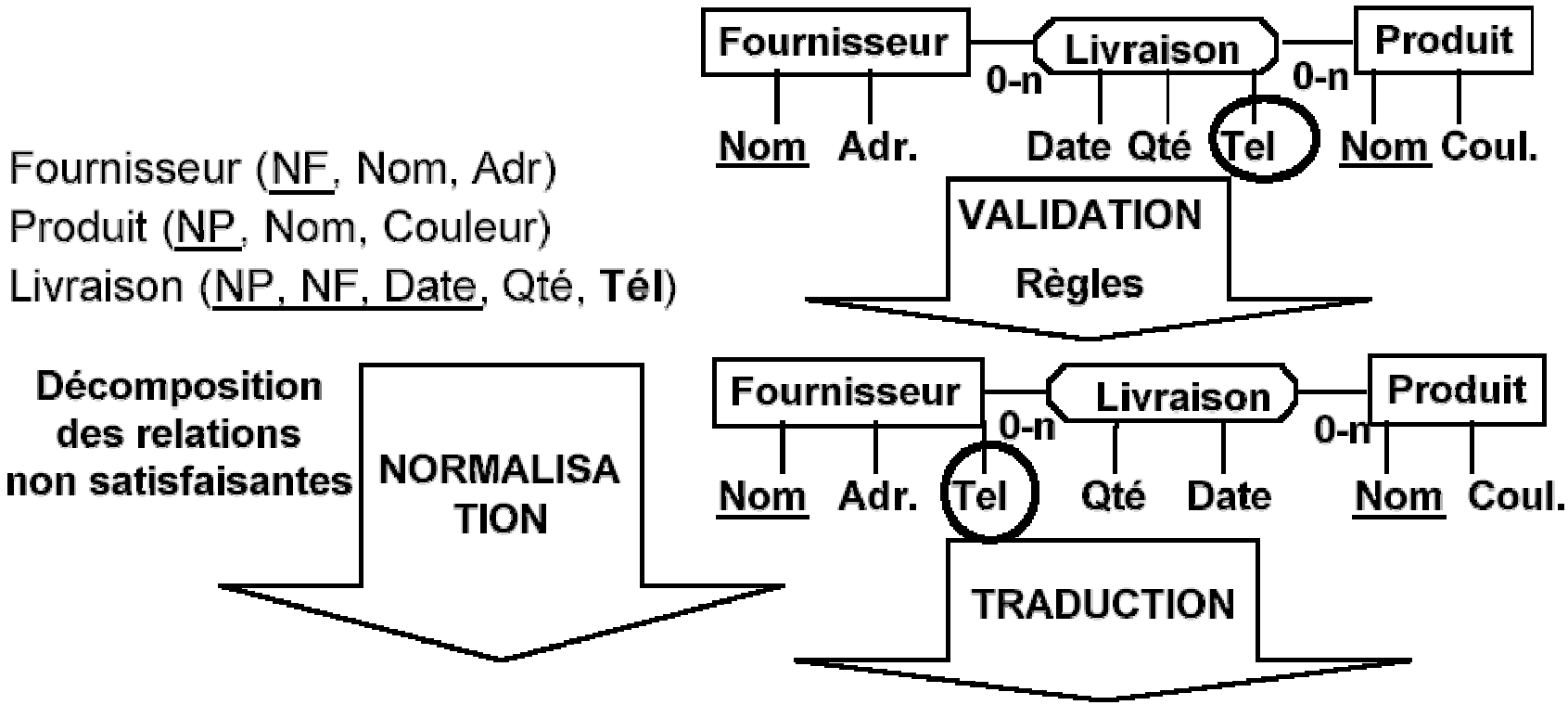
Normalisation

- Processus de transformation d'une relation posant des problèmes lors des MAJ en relations ne posant pas de problèmes
- On mesure la qualité d'une relation par son degré de normalisation
- 1NF, 2NF, 3NF, BCNF, 4NF, etc.

Normalisation ou traduction EA



Exemple


Fournisseur
Livraison
Produit
0-n
0-n
Nom
Adr.
Date
Qté
Tel
Nom
Coul.
VALIDATION
Règles
Fournisseur (NF, Nom, Adr)
Produit (NP, Nom, Couleur)
Livraison (NP, NF, Date, Qté, Tél)
Décomposition des relations non satisfaisantes
NORMALISATION
Fournisseur
Livraison
Produit
0-n
0-n
Nom
Adr.
Tel
Qté
Date
Nom
Coul.
TRADUCTION
Produit (NP, Nom, Couleur)
Fournisseur (NF, Nom, Adr, Tél)
Livraisonbis (NP, NF, Date, Qté)

Formalisation du problème

- L'adresse d'un fournisseur ne dépend que du fournisseur, ...

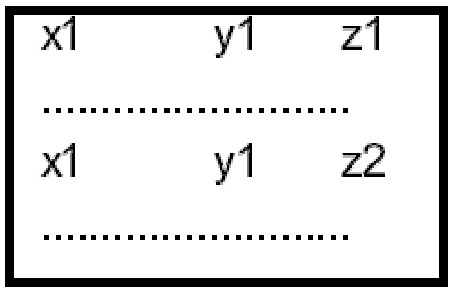
⇒ DÉPENDANCE FONCTIONNELLE (DF)

- soit une table T (x, y, z)

- il existe une DF: x → y si et seulement si dans T à une même valeur de x correspond toujours une même valeur de y

Formalisation

- T : X Y Z



- X → Y : X détermine Y Y dépend de X
- X: source de la DF, Y: cible de la DF
- la source peut être un ensemble d'attributs: (nom, prénom) → adresse

Propriétés des DF

- Transitivité:
- si X → Y et Y → Z alors X → Z

  →(DF déduite)
- Augmentation:
- si X → Y alors (A, X) → Y quelque soit A

  →(DF non élémentaire)
- On ne s'intéresse qu'aux DF élémentaires non déduites.

Graphe des DF

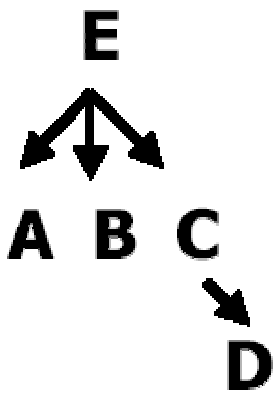
• Pour chaque table il faut connaître les DF intéressantes. Il est facile alors de les représenter sous forme de graphe:

graphe minimum des DF (orienté)

• Exemple: T (A, B, C, D, E)

E → A, E → B, E → C *(≡ E → A, B, C)*

C → D

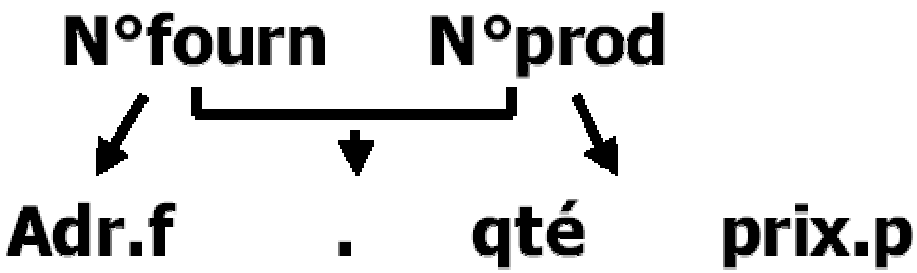


Exemple de graphe des DF

Livraison (N°fourn, adr.f, N°prod, prix.p, qté)

- N°fourn → adr.f
- l 'adresse d 'un fournisseur ne dépend que du fournisseur
- N°prod → prix-p
  - le prix d 'un produit ne dépend que du produit
- (N°fourn, N°prod) → qté
  - la quantité livrée dépend du produit **et** du fournisseur

[faux: *N°fourn → qté , N°prod → qté*]



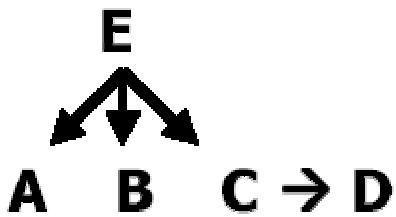
DFs et identifiants

- Le graphe minimum des DF permet de trouver les identifiants de la table

- L'identifiant d 'une table est l 'ensemble (minimal) des nœuds du graphe minimum à partir desquels on peut atteindre tous les autres nœuds (via les DF)

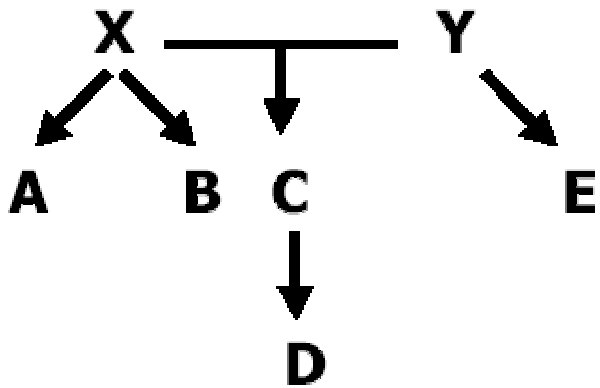
- Pour que ce soit faux il faudrait qu'il y ait deux lignes avec la même valeur de l'« identifiant » et des valeurs différentes pour les autres attributs, ce qui est en contradiction avec les DF.

Exemple : T1 (A, B, C, D, E)



DFs et identifiants

- Autre exemple: T2 (A, B, C, D, E, X, Y)



T2 (A, B, C, D, E, X, Y)

Normalisation

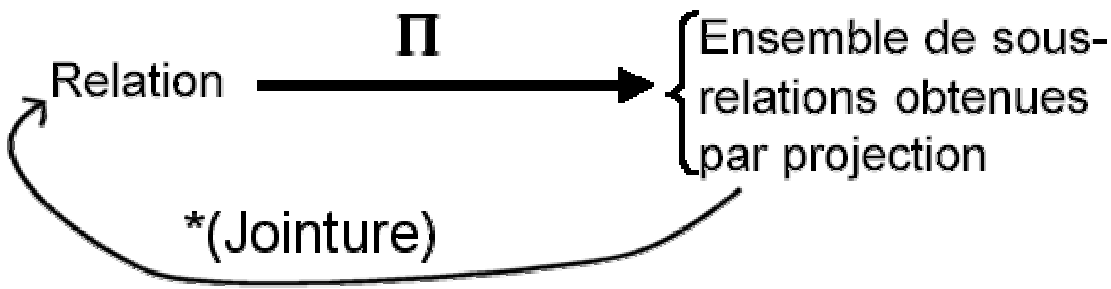
- Que faire si une table n'est pas « normalisée » ?

  ⇒ DECOMPOSITION

- La table doit être remplacée par un ensemble de tables (plus petites: moins d'attributs)

Décomposition d'une relation

- Soit une relation non satisfaisante, trouver un ensemble de relations satisfaisantes qui décrive les mêmes informations
- Pour vérifier qu'une relation est décomposable sans perte d'information:



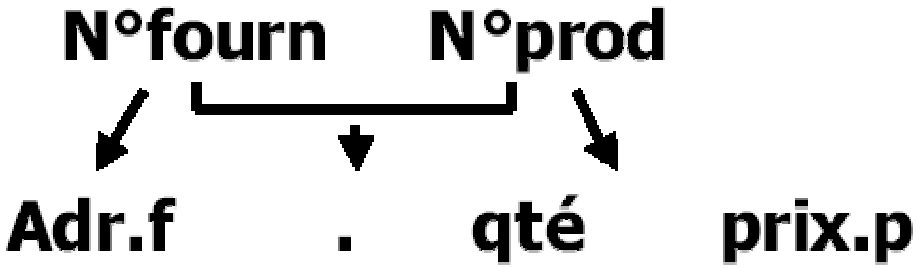
- Nombreux algorithmes de décomposition

Mais attention: les relations sont normalisées, mais peuvent être sémantiquement non significatives.

Méthode pragmatique

Livraison (N°fourn, adr.f, N°prod, prix.p, qté)

- Graphe des DF:



- Identifiant: N°fourn + N°prod
- Décomposition:
  - N°prod → prix.p ⇨ Prod (N°prod, prix.p)
  - N°fourn → adr.f ⇨ Fourn (N°fourn, adr.f)
  - (N°fourn, N°prod) → qté ⇨ Livr. (N°fourn, N°prod, qté)

Méthode formelle de décomposition

- Théorème de Heath

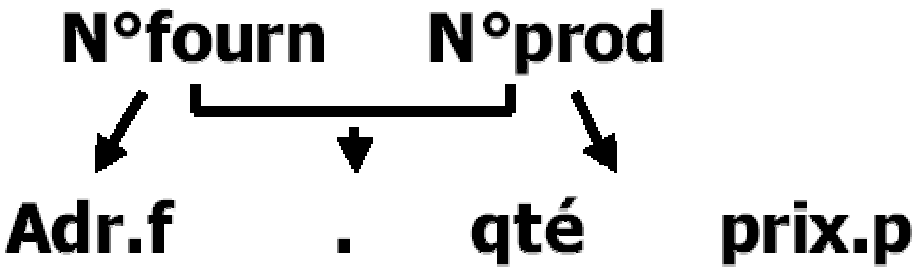
T (x, y, z) est décomposable sans perte d'information en

T1 (x, y) et T2 (x, z)

si x → y

Application de Heath

Livraison (N°fourn, adr.f, N°prod, prix.p, qté)



- N°fourn → adr.f. ⇨ (N°fourn, adr.f.) **ok**

  (N°fourn, N°prod, prix.p., qté)

- N°prod → prix.p ⇨ (N°prod, prix.p) **ok**

  (N°prod, N°fourn, qté)

- N°prod, N° fourn → qté **ok**

Qualité d'une décomposition

- Une « bonne » décomposition est une décomposition

  1) sans perte d 'information

  2) sans perte de DF

- Sans perte de DF:

toute DF doit être dans l'une des tables obtenues par décomposition

Sans perte d'information

- La « recomposition » de la table à partir des « morceaux » doit redonner la table initiale

- soit T (x, y, z) décomposée en T1 (x, y) T2 (x, z) tel que:

  T1 = PROJECTION [x, y] T

  T2 = PROJECTION [X, Z] T

- la décomposition est sans perte d 'informations si et seulement si :

  T = JOINTURE (T1, T2)

Exemple: bonne décomposition

- T (NomEmp, adresse, poste, age)

Zoé	Lausanne	secrétaire	27
Armand	Genève	secrétaire	32
Marie	Bienne	directeur	38

T1 (NomEmp, adresse, poste)

Zoé	Lausanne	secrétaire
Armand	Genève	secrétaire
Marie	Bienne	directeur

T2 (NomEmp, age)

Zoé	27
Armand	32
Marie	38

- **T = JOINTURE (T1, T2)**

Exemple: mauvaise décomposition

- T12 (NomEmp, adresse, poste) T22 (poste, age)

Zoé	Lausanne	secrétaire
Armand	Genève	secrétaire
Marie	Bienne	directeur

secrétaire	27
secrétaire	32
directeur	38

JOINTURE (T12, T22) =

Zoé	Lausanne	secrétaire	27
Zoé	Lausanne	secrétaire	32
Armand	Genève	secrétaire	27
Armand	Genève	secrétaire	32
Marie	Bienne	directeur	38

≠ T

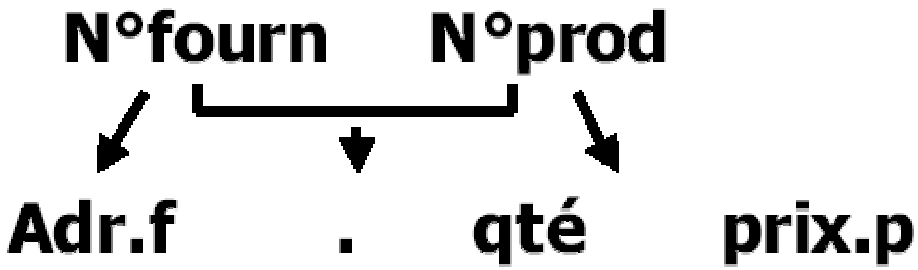
Formes normales: 1FN

- Une relation est en 1FN si chaque valeur de chaque attribut de chaque tuple est une valeur simple (tous les attributs sont simples et monovalués).
- Exemple:

  Livraison (N°fourn, adr.f, N°prod, prix.p, qté)

2ème forme normale: 2FN

- Permet d'éliminer les attributs qui ne décrivent pas l'« objet » traduit par la relation

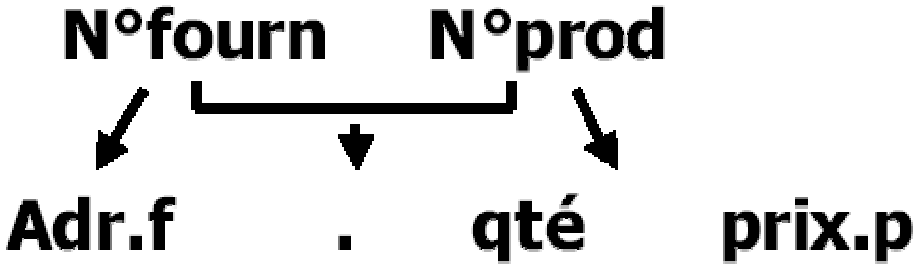


- mélange la description:

  - de la livraison (≠fourn., ≠produit, quantité-livrée)
  - du fournisseur (≠fourn., adr.-fourn)
  - du produit (≠produit, prix)

2ème forme normale: définition

Livraison (N°fourn, adr.f, N°prod, prix.p, qté)



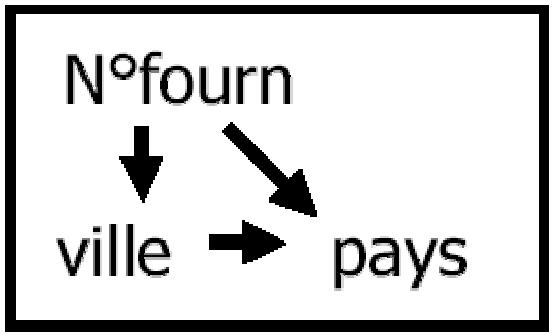
- des DF partent de composants de l 'identifiant: Livraison n'est pas en 2FN
- une table est en 2FN si :

  - elle est en 1FN, et

  - chaque attribut qui ne fait pas partie de l'identifiant dépend d'un identifiant **entier**

3FN: 3ème forme normale

- Permet ‘éliminer des sous-relations incluses dans une relation
- Exemple: Fournisseur (N°fourn, ville, pays)



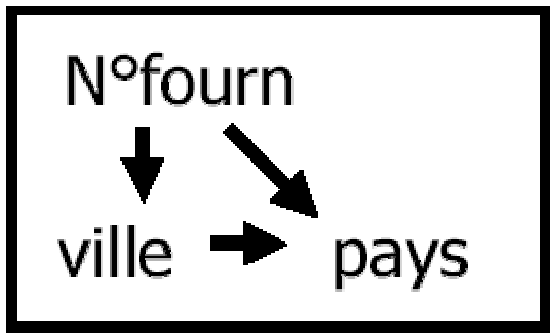
doit être décomposée en

F (N° fourn, ville)

G (ville, pays)

3ème forme normale: définition

- Fournisseur (N°fourn., ville, pays)



- Profondeur de l 'arbre des DF > 1 :

  pas en 3FN

- 3FN: si T est en 2FN et chaque attribut qui ne fait pas partie de l'identifiant dépend **directement** d'un identifiant entier

FNBC Forme normale de Boyce-Codd

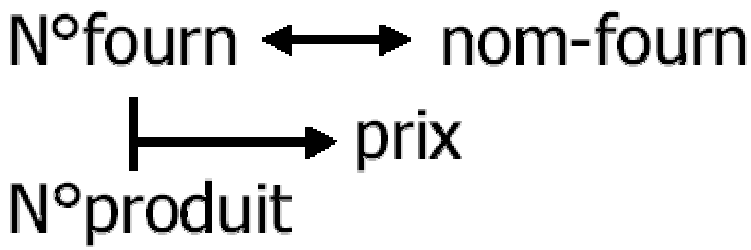
- Généralise la 3FN aux relations avec plusieurs identifiants

Fournisseur (N°fourn, nom-fourn, N°produit, prix)

- Une table est en FNBC si elle est en 3FN et si toute source complète de DF est un identifiant entier (BCNF)

FNBC: exemple Fournisseur

- Fournisseur (N°fourn, nom-fourn, N°produit, prix)

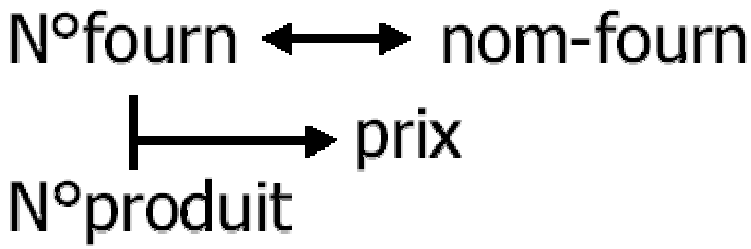


est en 3FN mais pas en FNBC

- le passage en FNBC n'est pas toujours possible sans perte de dépendances

Décompostion Fournisseur

- Fournisseur (N°fourn, nom-fourn, N°produit, prix)

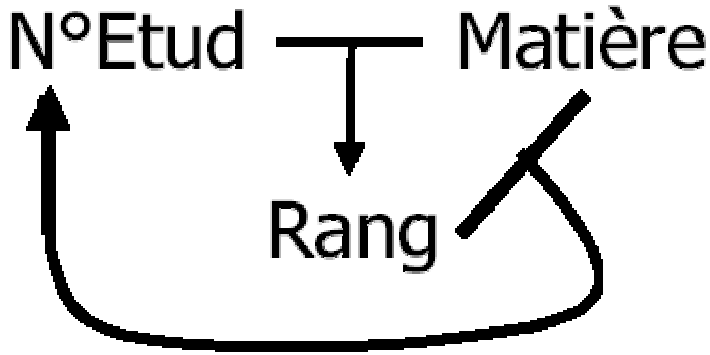


- F1 (N°fourn, nom-fourn)
- F2 (N°fourn, N°produit, prix) ou

F2' (nom-fourn, N°produit, prix)

FNBC: exemple Place

- Place (N°Etud, Matière, Rang)



- Identifiants :

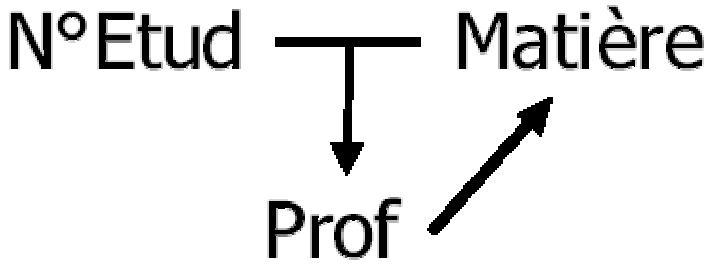
  1) N°Etud + Matière

  2) Rang + Matière

- Place est en 3FN et est en FNBC

FNBC contre-exemple

- Enseignement (N°Etud, Matière, Prof)



- Identifiants :

  1) N°Etud + Matière

  2) N°Etud + Prof

- Enseignement est en 3FN mais n'est pas en FNBC

Décomposition Enseignement

- Enseignement (N°Etud, Matière, Prof)

- Soit la décomposition:

  T1 (Prof, Matière) T2 (N°Etud, Prof)

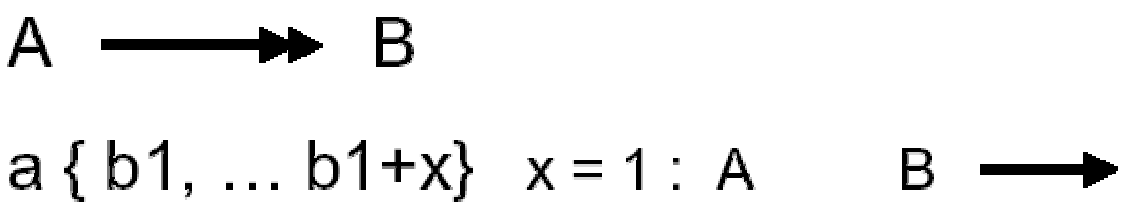
- T1 + T2 : sans perte d 'information mais avec perte de la DF:
(N°Etud, Matière) → Prof

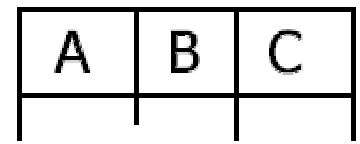
- il faut ajouter la C.I.:

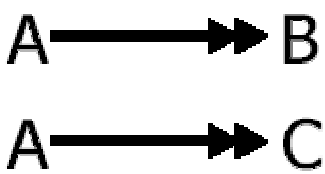
  un étudiant suit une matière donnée avec un seul professeur

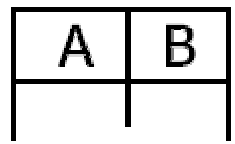
Quatrième Forme Normale (4FN)

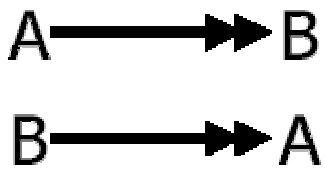
- Dépendance multivaluée (DM)





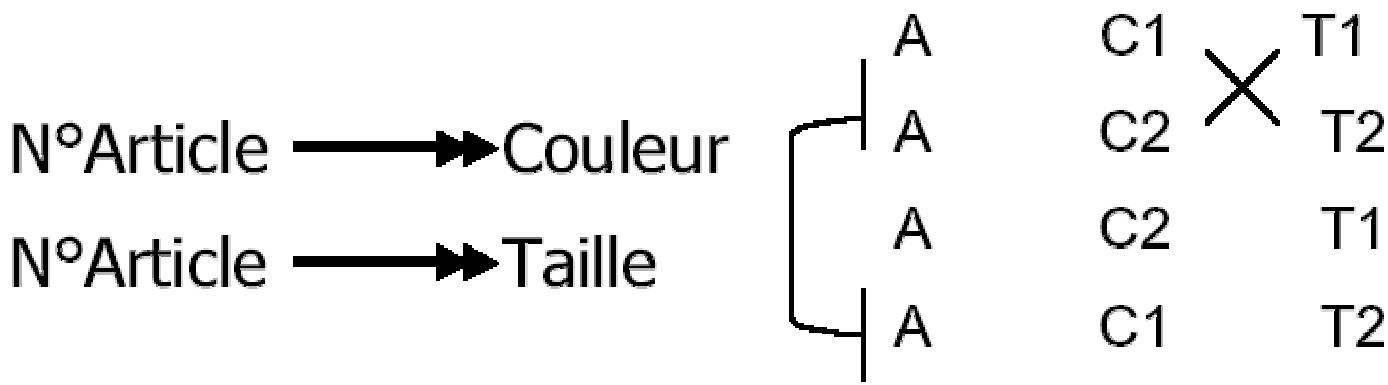






4FN: example et décomposition

Catalogue (N°Article, Couleur, Taille)



Décomposition : R1 (N°Article, Couleur)

R2 (N°Article, Taille)

(Heath)

4 FN: sémantique

- permet de séparer des fait indépendants qui auraient été réunis dans une même relation
- exemple:
  - il existe plusieurs cours
  - tout cours peut être assuré par plusieurs professeurs
  - tout cours possède un ensemble de livres de référence
- alors

cours (intitulé, nom-prof., titre-livre)

- donne une représentation très redondante, qu ’il convient de remplacer par décomposition

4 FN: définition

- cours (intitulé, nom-prof., titre-livre)

- décomposition:

  cours-p (intitulé, nom-prof.)

  cours-1 (intitulé, titre-livre)

- Formalisation: dans cours (intitulé, nom-prof., titre-livre) il existe une dépendance multivaluée intitulé ⟶ nom-prof, titre-livre

- Une relation est en 4FN s 'il n 'existe pas de dépendance multivaluée qui ne soit pas une dépendance fonctionnelle

4FN: autres définitions

- Soit une relation R (x,y,z) z pouvant être vide
- Il y a Dépendance multivaluée (DM)

- si à toute valeur de x correspond un ensemble de valeurs de y qui est totalement indépendant de z

Remarque: DF est un cas particulier de DM

- R est en 4 FN si elle est en 1ère FN et si toute DF ou DM de R a pour source un identifiant entier de R

Remarque: 4 FN implique BCNF (FNBC)

Cinquième Forme Normale (5FN)

- Permet de décomposer une relation en faits élémentaires
- exemple: Vins (buveur, cru, producteur)
- Peut être décomposable en

  V1 (buveur, cru)

  V2 (buveur, producteur)

  V3 (cru, producteur)

  Avec Vins = V1 |X| V2 |X| V3
- On dit qu'il y a dépendance de jointure

Conception d'une BD relationnelle

(niveau conceptuel)

- Méthode classique
- Objectif: arriver à un ensemble de relations tel que

  - chaque relation décrit un type d' « objet » avec les seuls attributs qui lui sont directement liés

  - les redondances d 'information (génératrices de problèmes lors des mises à jour) sont éliminées

Conception d'une BD relationnelle

- Méthode: Décomposition par projections sans perte d 'information et sans perte de dépendances
- Outils:
  - dépendances (fonctionnelles, multivaluées, de jointure)
  - règles de normalisation
- Limites:
  - ne garantit pas que le schéma soit celui qu 'il faut
  - ne garantit pas que le schéma soit efficace
- (« overnormalization »)

Conception d'une BD relationnelle

- Méthode « moderne »

- Conception d'un schéma entité-association, puis traduction de celui-ci en schéma relationnel