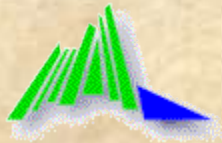


Bases de Données
Langage SQL

Manuel Munier

IUT des Pays de l'Adour - Mont de Marsan

Département Réseaux Télécommunications

2011-2012

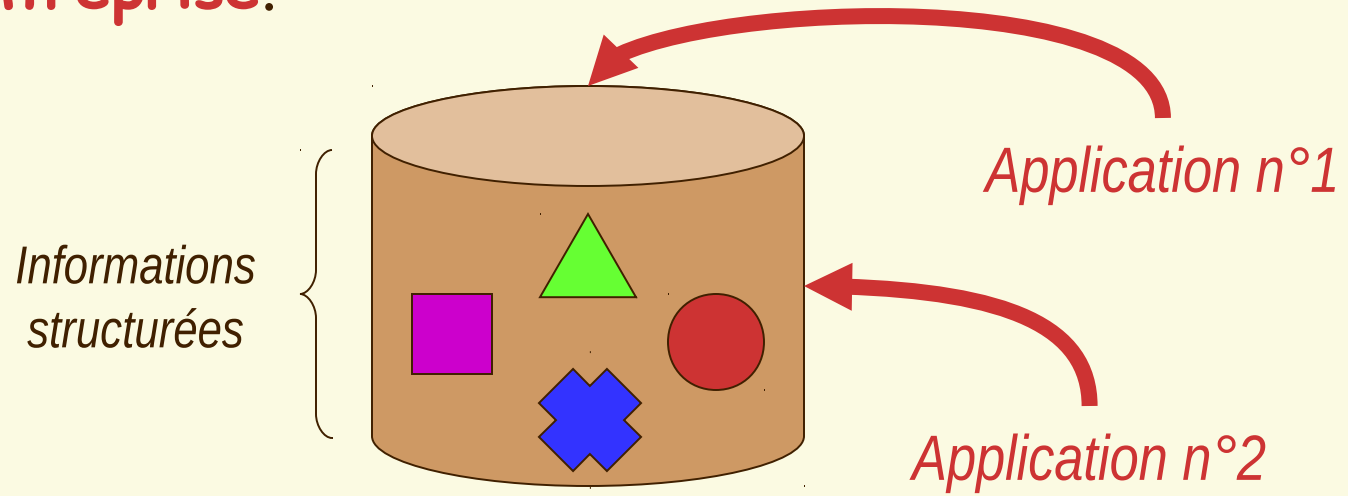
Plan

BdD

- Introduction
- Un survol des bases de données
- Exploitation d'une base de données
  - algèbre relationnelle (un peu…)
  - un langage de requêtes: SQL
- Conception d'une base de données
  - modèle entités-associations
  - modèle relationnel

Introduction

- Définition informelle d'une BdD:
  - Une **base de données** (BdD) est un ensemble **structuré** d'informations **persistantes** partagées par plusieurs **applications** d'une même **entreprise**.



Définition d'une BdD

- Ensemble structuré
  - les données ont une structure qui a été définie une fois pour toutes
  - cette structure doit donc être définie en fonction de l'exploitation ultérieure de ces données
- Plusieurs applications
  - les utilisateurs, au travers de plusieurs applications, se partagent ces données mais peuvent avoir des préoccupations différentes

Définition d'une BdD

- Entreprise
  - les applications ne sont pas indépendantes: elles appartiennent à la même entreprise (au sens large: université, banque, PME/PMI,…)
- Informations persistantes
  - les données sont conservées de manière permanente (persistance) et elles sont disponibles pour chaque application, sans qu'il y ait besoin de les réintroduire dans le système

Exemple

- Données
  - liste des étudiants inscrits,
  - liste des cours,
  - liste des enseignants,
  - emplois du temps,
  - relevés de notes,…
- Applications
  - gestion des inscriptions,
  - planning des salles,
  - jurys d'examens,…
- Entreprise
  - université

Exploitation d'une BdD

- On peut imaginer que les données sont stockées dans des fichiers (ou des tableaux)
- Mais les informations stockées ne sont pas les seules informations accessibles !
  - « Pourcel suit le cours de BdD en RT »
  - « Munier est l'enseignant de BdD en RT »
- On peut en déduire l'information
  - « Munier enseigne les BdD à Pourcel »

Exploitation d'une BdD

- Une base de données contient à la fois
  - des informations représentant des objets du monde extérieur
  - des liens sémantiques entre ces objets
- Exploiter une BdD, c'est savoir
  - insérer de nouvelles informations
  - extraire, parmi toutes ces informations, celles dont on a besoin
  - manipuler les relations entre ces informations

Objectifs d'une BdD

- Une approche BdD nous apporte
  - intégration
    - toutes les données d'une entreprise sont placées dans un référentiel commun à toutes les applications qui y puisent les données les concernant
  - flexibilité
    - données indépendantes d'une application particulière
    - SGBD ⇒ indépendance vis-à-vis du support physique
  - disponibilité
    - persistance, performances du serveur, réseau,…
  - sécurité
    - pannes, confidentialité, accès concurrents,…

Plan

- Introduction
- Un survol des bases de données
- Exploitation d'une base de données
  - algèbre relationnelle (un peu…)
  - un langage de requêtes: SQL
- Conception d'une base de données
  - modèle entités-associations
  - modèle relationnel

Objectifs

Survol

- Pourquoi commencer par un survol ?
  - 1er TP après 1h30 de cours et 1h30 de TD
  - dès le 1er TP il faut savoir interroger une BdD
- Que va-t-on voir ?
  - comment sont structurées les données (niveau conceptuel)
  - comment sont-elles représentées dans la BdD (niveau physique: tables)
  - comment exploiter une BdD à l'aide d'un langage de requêtes (SQL)

Conception d'une BdD

Survol

- Formalisme utilisé: modèle entités-associations (dû à Chen)
  - ou *modèle individus-relations*
  - ou *modèle individuel*
- But: décrire le réel perçu à l'aide de ce formalisme
  - à partir d'un cahier des charges
  - par analyse d'un système existant
  - etc…

Modèle E-A: Entités

Survol

- Une **entité** est un objet physique ou abstrait ayant une existence propre et pouvant être différencié par rapport aux autres objets.
  - objets physiques
    - personne, voiture,…
  - objet abstrait
    - « vol #IJ509 pour Metz partant de Bordeaux »
  - contre-exemple
    - un « livre de BdD écrit en 1975 » n'est pas une entité car il n'est pas unique

Modèle E-A: Entités

Survol

- Une entité est décrite par l'ensemble de ses propriétés appelées **attributs**.
  - Ne pas confondre le nom de l'attribut et sa valeur
    - `NumVol` ⇒ nom
    - `#IJ509` ⇒ valeur
  - Les valeurs doivent appartenir à un ensemble de valeurs (domaine)
    - `D1 = {Pourcel,Cousy,Burgy} = NomEtudiant`
    - `D2 = {BdD,Java,SE,Réseaux} = NomCours`
    - `D3 = {x | x ∈ [0..20]} = Note`

Modèle E-A: Entités

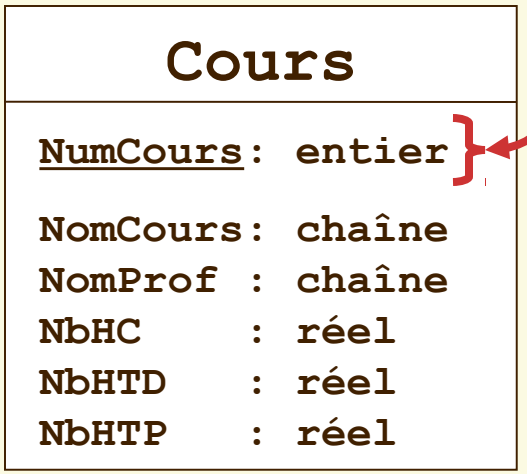
Survol

- Les trois entités suivantes ont la même structure (i.e. les mêmes attributs)
  - **(C1,Java,Munier,12,10.5,15)**
  - **(C2,Réseaux,Bascou,24,30,30)**
  - **(C3,TransNum,Baillot,12,18,24)**
- On dit qu'elles appartiennent à une même classe d'entités ou **type d'entité**
- Ici, la classe **Cours** est caractérisée par ses six attributs **NumCours**, **NomCours**, **NomProf**, **NbHC**, **NbHTD** et **NbHTP**

Modèle E-A: Entités

Survol

- Représentation graphique du type d'entité **Cours**



Identifiant (ou **clé**) = attribut(s) permettant d'identifier de manière unique une entité

En effet, l'attribut **NomCours** ne suffit pas (ex: **SE** en RT1 et **SE** en RT2)

Si aucun attribut ne convient, il suffit de créer artificiellement un identifiant unique pour chaque entité

Modèle E-A: Entités

Survol

- Type d'entité **Etudiant**

Etudiant
NumEtud : entier
Nom : chaîne
Prenom : chaîne
Adresse : chaîne
DateNais: date
Sexe : {M,F}

Le couple d'attributs **(Nom,Prenom)** ne peut pas servir de clé car ce n'est pas suffisant pour identifier de manière unique un étudiant (cas des homonymes).

⇒ On crée un attribut **NumEtud** qui servira d'identifiant.

Modèle E-A: Entités

Survol

- Au niveau physique, une type d'entité sera représenté par une table
  - chaque **colonne** correspond à un **attribut**
  - chaque **ligne** représente une **entité** de ce type

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Modèle E-A: Entités

Survol

- Certains liens entre les informations n'existent pas encore
  - lien entre les entités **`Cours`** et **`Etudiant`** pour indiquer que tel étudiant suit tel cours
- Il nous faut donc enrichir ce modèle pour prendre en compte ces informations mettant en relation plusieurs entités

Modèle E-A: Associations

Survol

- Une **association d'entités** est un regroupement de deux ou plusieurs entités pour décrire une réalité de l'organisation
- Soit les deux entités suivantes
  - `(C1,Java,Munier,12,10.5,15)`
  - `(E1,Cousy,Cécile,?,?,F)`
- L'association ci-dessous exprime le fait que l'étudiant(e) `E1` a suivi le cours `C1` et a obtenu la note de 18.5
  - `(E1,C1,18.5)`

Modèle E-A: Associations

Survol

- Un type d'association (d'entités) est un sous-ensemble d'un produit cartésien d'entités. Il permet de représenter les informations n'ayant de sens que par rapport à l'association de certains types d'entités.
- Bref, c'est un **lien sémantique** entre plusieurs types d'entités.

Modèle E-A: Associations

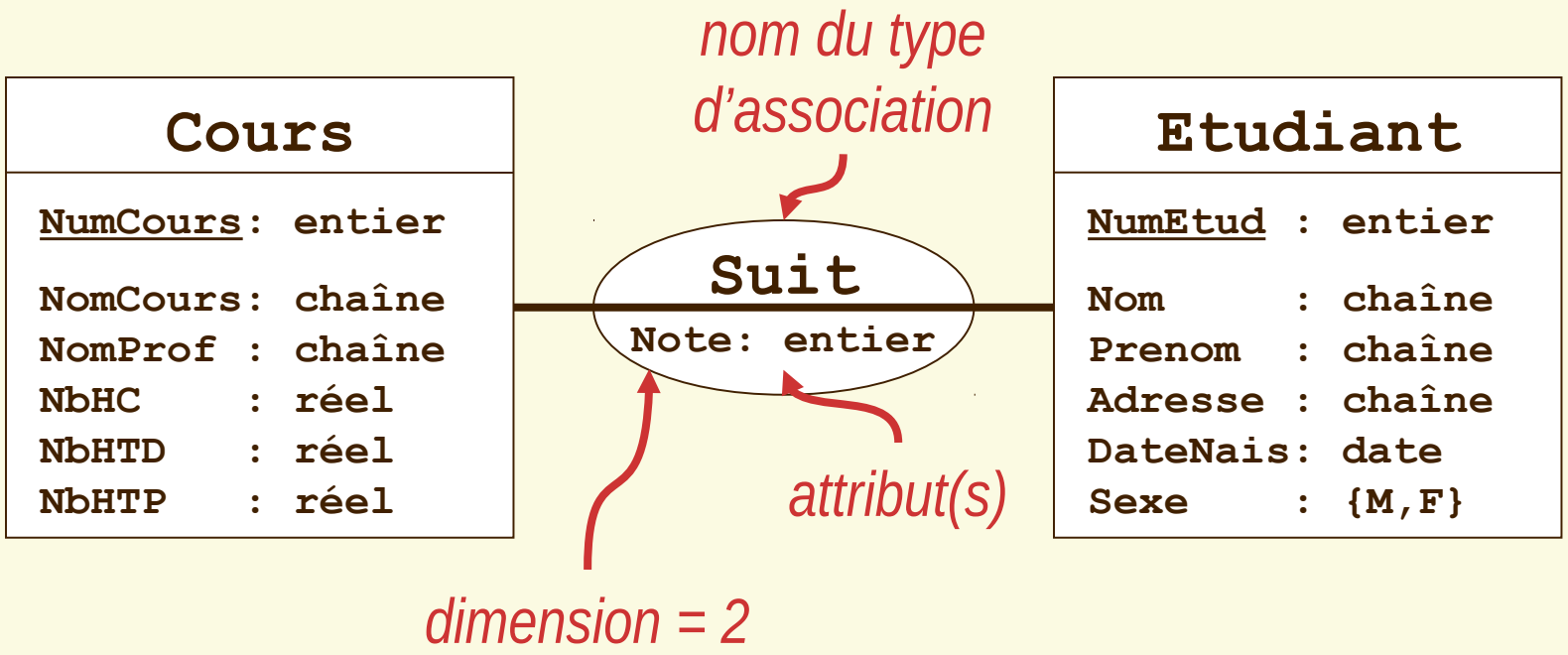
Survol

- Propriétés d'un type d'association
  - attributs
    - au minimum les identifiants des types d'entités reliés
    - éventuellement des attributs spécifiques (ex: **`Note`**)
  - identifiant
    - concaténation des identifiants des types d'entités reliés (ici, le couple **`(NumCours,NumEtud)`**)
    - NB: les identifiants des types d'association doivent eux aussi être uniques
  - dimension
    - nombre de types d'entités reliés (généralement 2)

Modèle E-A: Associations

Survol

- Représentation graphique du type d'association **Suit**



Modèle E-A: Associations

Survol

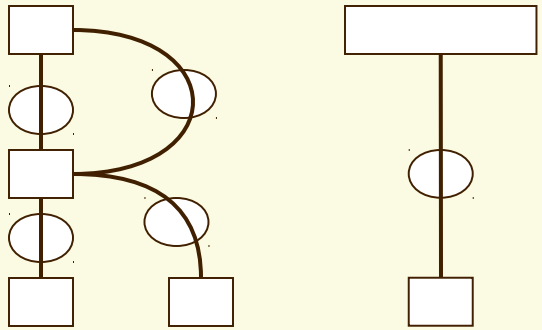
- Au niveau physique, une type d'association sera également représenté par une table
  - chaque **colonne** correspond à un **attribut**
  - chaque **ligne** représente une **association** de ce type entre deux entités

Suit	NumEtud	NumCours	Note
	E1	C1	18,5
	E1	C2	15
	E2	C1	9,5

Modèle E-A

Survol

- On reviendra sur le modèle entités-associations à la fin du cours
  - notations supplémentaires
  - extensions au modèle E-A
  - comment le concevoir intelligemment
  - optimisations (limiter les redondances,…)



Exploitation d'une BdD

Survol

- Pour accéder à une base de données, on distingue deux outils
  - LDD (langage de définition des données)
    - création du schéma (tables, index,…)
    - gestion des droits d'accès
  - LMD (langage de manipulation des données)
    - manipulation des informations
      - insertion
      - modification
      - suppression
    - recherche d'informations
    - statistiques sur ces informations

Langage SQL

Survol

- **SQL** = Structured Query Language
  - langage de requête structuré
  - conçu par IBM dans les années 70
  - norme SQL2 définie en 1992
- SQL est à la fois un LDD et un LMD
- Dans ce survol, on va se contenter de voir comment interroger une BdD à l'aide de l'instruction `select` (forme simplifiée)

Instruction select

Survol

- Construction de base d'une requête SQL

select $a_1,\ldots,a_p$
from $T_1,\ldots,T_n$
where B

- Avec
  - les a_i sont des attributs et représentent le résultat attendu de la requête
  - les T_i indiquent quelles sont les tables concernées par cette requête (où vont être récupérées les informations)
  - B est une condition booléenne sur les a_i

Instruction select

Survol

- Remarques
  - on peut éliminer les tuples en double en faisant précéder la liste des attributs citée dans le **select** par le mot-clé **distinct**

```
select distinct a1,…,ap
from T1,…,Tn
where B
```

  - * est une convention qui remplace tous les attributs des tables citées dans le **from**
  - le **where** est facultatif

Instruction select

Survol

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Suit

NumEtud	NumCours	Note
E1	C1	18,5
E1	C2	15
E2	C1	9,5

Instruction select

Survol

- Exemple

```
select NumEtud,Nom,Prenom,Sexe
from Etudiant
where Sexe=M
```

- Résultat

```
NumEtud   Nom       Prenom    Sexe
--------  -------   -------   -----
E2        Pourcel   Mathieu   M
E3        Burgy     Laurent   M

2 ligne(s) selectionnee(s)
```

Instruction select

Survol

Etudiant	NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
	E1	Cousy	Cécile	?	?	F
	E2	Pourcel	Mathieu	?	?	M
	E3	Burgy	Laurent	?	?	M

Cours	NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
	C1	Java	Munier	12	10,5	15
	C2	Réseaux	Bascou	24	30	30
	C3	TransNum	Baillot	12	18	24

Suit	NumEtud	NumCours	Note
	E1	C1	18,5
	E1	C2	15
	E2	C1	9,5

Instruction select

Survol

- Exemple

```
select Nom,Prenom,NomCours,NomProf,Note
from Etudiant,Cours,Suit
where (Suit.NumEtud = Etudiant.NumEtud)
  and (Suit.NumCours = Cours.NumCours)
```

- Résultat

```
Nom      Prenom   NomCours  NomProf   Note
-------  -------- --------- --------- -----
Cousy    Cécile   Java      Munier    18.5
Cousy    Cécile   Réseaux   Bascou    15
Pourcel  Mathieu  Java      Munier    9.5

3 ligne(s) selectionnee(s)
```

Instruction select

Survol

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Suit

NumEtud	NumCours	Note
E1	C1	18,5
E1	C2	15
E2	C1	9,5

Instruction select

Survol

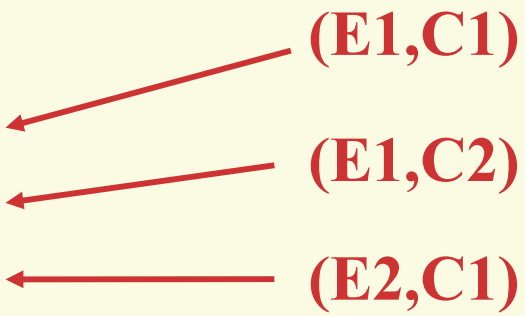
- Exemple

```
select NomCours,NomProf
from Cours,Suit
where (Suit.NumCours = Cours.NumCours)
```

- Résultat

```
NomCours  NomProf
--------- ---------
Java      Munier
Réseaux   Bascou
Java      Munier

3 ligne(s) selectionnee(s)
```



Instruction select

Survol

- Exemple

```
select distinct NomCours,NomProf
from Cours,Suit
where (Suit.NumCours = Cours.NumCours)
```

- Résultat

```
NomCours  NomProf
--------- ---------
Java      Munier
Réseaux   Bascou

2 ligne(s) selectionnee(s)
```

distinct supprime les doublons

Instruction `select`

Survol

- Opérateurs de comparaison

```
=  >=  >  !=  <  <=
between … and …
in
is null
is not null
like …
```

- Opérateurs logiques

```
not
and
or
```

Instruction select

Survol

- Exemples de conditions
  - La note est comprise entre 8 et 16

    ```
    Note between 8 and 16
    ```

  - Le nom du cours est Java, BdD ou Réseaux

    ```
    NomCours in ('Java','BdD','Réseaux')
    ```

  - Le prénom commence par la lettre C

    ```
    Prenom like 'C%'
    ```

Instruction select

Survol

- On peut trier les tuples retournés par une requête en ajoutant une clause **order by**
  - recherche tous les cours réalisés par Munier ou Gallon et les affiche par ordre croissant de **NbHC**, puis **NbHTD**, puis **NbHTP**

```
select *
from Cours
where NomProf in ('Munier','Gallon')
order by NbHC,NbHTD,NbHTP
```

Instruction select

Survol

- Le tri peut se faire
  - par ordre croissant (**ASC**, par défaut)
  - par ordre décroissant (**DESC**)

```
rem Classement au partiel de Java
select Nom,Prenom,Note
from Suit,Etudiant,Cours
where (Suit.NumEtud = Etudiant.NumEtud)
  and (Suit.NumCours = Cours.NumCours)
  and (NomCours = 'Java')
order by Note DESC
```

Conclusion

Survol

- On a vu le strict minimum sur les BdD pour avoir une idée de leur fonctionnement
  - pourquoi utiliser des BdD
  - comment les concevoir
    - schéma entités-associations
    - notion d'identifiant (clé)
    - entités et associations sont traduites en tables
  - comment les exploiter
    - un langage de requêtes: SQL
    - forme de base d'une requête SQL (le `select`)

Plan

- Introduction
- Un survol des bases de données
- Exploitation d'une base de données
  - algèbre relationnelle (un peu…)
  - un langage de requêtes: SQL
- Conception d'une base de données
  - modèle entités-associations
  - modèle relationnel

Algèbre relationnelle

- Dans une BdD
  - les données sont rangées dans des tables
  - une requête sur la base est un algorithme dont les paramètres sont des tables de la base
  - le résultat d'une requête est également une table
- Idée:
  - pouvoir utiliser la table résultant d'une requête comme paramètre d'un autre algorithme d'interrogation

Algèbre relationnelle

- Objectif
  - définir un certain nombre d'opérations élémentaires sur les tables de façon à ce qu'une requête quelconque puisse s'exprimer en combinant ces opérations
- En algèbre relationnelle, une **requête** est une expression formée
  - de **variables** (les tables de la base)
  - de **constantes**
  - d'**opérateurs** (les opérateurs de l'algèbre rel.)

Relations

Algèbre

- Définition
  - soient $D_1, \ldots, D_n$ des ensembles de valeurs non nécessairement disjoints
  - une **relation** (n-aire) R définie sur $D_1, \ldots, D_n$ est un sous-ensemble du produit cartésien $D_1 \times \ldots \times D_n$
- Une relation est un ensemble de **n-uplets** $<a_1, \ldots, a_n>$ où, pour chaque a_i, on a $a_i \in D_i$

Relations

Algèbre

- Plutôt que de désigner les colonnes par leur rang, on leur donne un nom; on parle alors d'**attributs**
- Exemple de descripteur de relation:

$$R(A_1 : dom(A_1), \ldots, A_n : dom(A_n))$$

- où:
  - **R** est le nom de la relation
  - les A_i sont les noms des attributs de la relation
  - les $dom(A_i)$ sont les domaines associés

Relations

- Quelques remarques
  - plusieurs attributs peuvent avoir le même domaine de valeurs
    - **ex: Vol(numVol: Numéro, départ: Ville, arrivée: Ville)**
  - les valeurs possibles d'un attribut sont supposées être des **valeurs atomiques** (i.e. non structurées)

Relations

- Quelques remarques
  - si $X=\{x_1,\ldots,x_i\}$ et $Y=\{y_1,\ldots,y_j\}$, alors $R(x_1,\ldots,x_i,y_1,\ldots,y_j)$ pourra également être noté $R(X,Y)$
  - on distinguera le **descripteur** d'une relation (défini une fois pour toute) du **contenu** de la relation (qui varie au cours du temps)
    - le descripteur sera noté R
    - l'ensemble des tuples sera noté r

Exemple de BdD

Algèbre

- Notre base de données **Magasin** contient
  - un ensemble d'attributs
    - **numFour, nomFour, remise, ville, numProd, nomProd, couleur, poids,origine, qte**
  - des relations sur ces attributs
    - **Fournisseur(numFour,nomFour,remise,ville)**
    - **Produit(numProd,nomProd,couleur,poids,origine)**
    - **Stock(numFour,numProd,qte)**

Exemple de BdD

Algèbre

Fournisseur			
numFour	**nomFour**	**remise**	**ville**
f1	Dupont	0	Paris
f2	Courvite	10	Marseille
f3	Frip64	5	Pau
f4	Alpages	3	Grenoble
f5	Stanislas	0	Nancy

Produit				
numProd	**nomProd**	**couleur**	**poids**	**origine**
p1	veste	bleu	0,3	Paris
p2	pantalon	noir	0,4	Lyon
p3	chemise	blanc	0,2	Londres
p4	veste longue	brun	0,6	Londres
p5	jean	bleu	0,5	Bordeaux
p6	manteau	rouge	1,2	Paris
p7	chemise	vert	0,2	Paris

Stock		
numFour	**numProd**	**qte**
f1	**p1**	300
f1	**p2**	200
f3	**p2**	200
f2	**p1**	300
f4	**p2**	200
f1	**p4**	200
f1	**p3**	400
f2	**p2**	400
f4	**p4**	300
f4	**p5**	400
f1	**p6**	100
f1	**p5**	100
f2	**p7**	150
f4	**p7**	100
f2	**p6**	50
f4	**p1**	200

Clé d'une relation

Algèbre

- La **clé** d'une relation est un **sous-ensemble minimal d'attributs** de la relation permettant d'identifier de manière unique un tuple de cette relation
- Exemples:
  - **{numFour}** est une clé pour **Fournisseur**
  - **{numProd}** est une clé pour **Produit**
  - **{numFour,numProd}** est une clé pour **Stock**

Opérateurs algébriques

Algèbre

- Opérateurs de base
  - projection
  - sélection
  - jointure naturelle (composition)
  - produit cartésien
- Opérateurs ensemblistes
  - union
  - intersection
  - différence

Projection

Algèbre

- Définition
  - Soit **R(Z)** une relation avec **Z=X,Y**. La **projection** de **R** sur **Y**, notée **R[Y]**, est définie par:

    <y>∈ r[Y] ssi **∃a** tel que **<a,y>∈ r**

- Intuitivement
  - On supprime les colonnes non retenues dans la projection et on élimine les tuples en double
- Exemple

  Stock[numFour]

NumFour
f1
f2
f3
f4

Sélection

- Définition
  - Soit **R(X)** une relation et **B(X)** un prédicat applicable à tout n-uplet de **R**. La **sélection** de **R** par **B**, notée **R{B}**, est définie par:

    <x>∈ r{B} ssi **<x>∈ r** et **B(x)** vaut vrai
- Intuitivement
  - On parcourt tous les tuples de la relation **R** et on ne garde que ceux qui vérifient le prédicat **B**

Sélection

- Exemples

Produit{origine='Paris'}

numProd	nomProd	couleur	poids	origine
p1	veste	bleu	0,3	Paris
p6	manteau	rouge	1,2	Paris
p7	chemise	vert	0,2	Paris

Fournisseur{remise=0}

numFour	nomFour	remise	ville
f1	Dupont	0	Paris
f5	Stanislas	0	Nancy

Jointure

Algèbre

- Définition
  - Soient **S(X,Z)** et **R(Z,Y)** deux relations. La **jointure** (naturelle) de **S** et de **R**, notée **S*R**, est définie par:

    <x,y,z>∈ S*R ssi <x,z>∈ s et <z,y>∈ r

- Intuitivement
  - Pour chaque tuple **<x,z>** de **S** on construit dans **S*R** autant de tuples **<x,y,z>** qu'il y a de tuples **<z,y>** dans **R**

Jointure

- Exemple

la jointure est réalisée sur cet attribut commun aux deux relations

Produit*Stock

nomProd	couleur	poids	origine	numProd	NumFour
veste	bleu	0,3	Paris	**p1**	**f1**
veste	bleu	0,3	Paris	**p1**	**f2**
veste	bleu	0,3	Paris	**p1**	**f4**
pantalon	noir	0,4	Lyon	**p2**	**f1**
pantalon	noir	0,4	Lyon	**p2**	**f2**
pantalon	noir	0,4	Lyon	**p2**	**f3**
pantalon	noir	0,4	Lyon	**p2**	**f4**
chemise	blanc	0,2	Londres	**p3**	**f1**
...	...	...	...	...	...

Jointure

- Exemple

Produit*Fournisseur

numProd	nomProd	couleur	poids	origine	numFour	nomFour	remise	ville
p1	veste	bleu	0,3	Paris	**f1**	Dupont	0	Paris
p1	veste	bleu	0,3	Paris	**f2**	Courvite	10	Marseille
p1	veste	bleu	0,3	Paris	**f3**	Frip64	5	Pau
p1	veste	bleu	0,3	Paris	**f4**	Alpages	3	Grenoble
p1	veste	bleu	0,3	Paris	**f5**	Stanislas	0	Nancy
p2	pantalon	noir	0,4	Lyon	**f1**	Dupont	0	Paris
p2	pantalon	noir	0,4	Lyon	**f2**	Courvite	10	Marseille
p2	pantalon	noir	0,4	Lyon	**f3**	Frip64	5	Pau
p2	pantalon	noir	0,4	Lyon	**f4**	Alpages	3	Grenoble
p2	pantalon	noir	0,4	Lyon	**f5**	Stanislas	0	Nancy
p3	chemise	blanc	0,2	Londres	**f1**	Dupont	0	Paris
...	...	...	...	...	...	...	...	...

Jointure

- **Attention:** s'il n'y a pas d'attribut commun entre les deux relations, la jointure calcule toutes les combinaisons possibles
- Exemple
  - une jointure incontrôlée entre les relations
    - EtudiantGTR (~110 tuples)
    - Cours (~30 tuples)
    - Salle (~12 tuples)
  - et on obtient 110x30x12 = 39600 tuples

Produit cartésien

Algèbre

- C'est une forme particulière de jointure
- Définition
  - Soient $S(X,Z)$ et $R(Z,Y)$ deux relations avec $X \cap Y = \emptyset$ et Z éventuellement vide. Le **produit cartésien** de S et de R, noté $S \times R$, est défini par:

    $\langle x, z_1, z_2, y \rangle \in S \times R$ ssi $\langle x, z_1 \rangle \in s$ et $\langle z_2, y \rangle \in r$
- Intuitivement
  - On associe à chaque tuple de S chacun des tuples de R

Opérateurs ensemblistes

Algèbre

- Définition
  - Soient **S(X)** et **R(X)** deux relations. Etant donné qu'elles sont deux issues du même ensemble (elles ont les mêmes attributs), on peut définir les opérateurs suivants:
    - union **S ∪ R**
    - intersection **S ∩ R**
    - différence **S - R**
- Exemple

(Stock*Produit{couleur='rouge'})[numFour]
∩ (Stock*Produit{couleur='bleu'})[numFour]

Exemples

Algèbre

- Voici quelques exemples d'interrogations:
  - Numéros des fournisseurs qui ont livré au moins un produit
    - `Stock[numFour]`
  - Numéros des fournisseurs qui n'ont livré aucun produit
    - `Fournisseur[numFour] - Stock[numFour]`
  - Numéros des fournisseurs qui ont livré le produit p2
    - `Stock{numProd='p2'}[numFour]`

Exemples

Algèbre

- Numéros des fournisseurs qui ont livré au moins un produit différent de p2
  - `Stock{numProd≠'p2'}[numFour]`
  - `Stock[numFour]-Stock{numProd='p2'}[numFour]`
- Numéros des fournisseurs qui n'ont livré que le produit p2
  - `Stock[numFour]-Stock{numProd≠'p2'}[numFour]`
- Fournisseurs qui ont livré au moins deux produits
  - S1 et S2 sont deux alias sur la relation Stock
  - `(S1×S2){S1.numFour=S2.numFour`

`∧ S1.numProd≠S2.numProd}[S1.numFour]`

Plan

- Introduction
- Un survol des bases de données
- Exploitation d'une base de données
  - algèbre relationnelle (un peu…)
  - un langage de requêtes: SQL
- Conception d'une base de données
  - modèle entités-associations
  - modèle relationnel

Création d'une table

SQL

- En SQL, chaque relation est représentée par une table
- Une table est créée à l'aide de la commande SQL **CREATE TABLE**

```
CREATE TABLE Stock (description)
```

- La description est la liste des attributs et des contraintes sur la table

Création d'une table

SQL

- Définition d'un attribut

identificateur type **[NULL|NOT NULL]**

- Types de données possibles avec Oracle
  - **CHAR(n)** : chaîne de caractères de longueur fixe n (maximum=255)
  - **VARCHAR(n)** : chaîne de caractères de longueur variable avec maximum n caractères (maxi=2000, défaut=1)
  - **LONG VARCHAR**: chaîne de caractères de longueur variable avec un maximum de 2Go

Création d'une table

SQL

- Types de données Oracle (suite)
  - **NUMBER(p,s)** : valeur numérique avec une précision de p chiffres dont s à droite du point décimal ($1 \leq p \leq 38$, défaut p=38, $s \in [-84,127]$)
  - **DATE** : date…
  - **RAW(n)** : chaîne de bits de taille maximum n octet(s) (maximum=2000)
  - **LONG RAW** : chaîne de bits avec une taille maximum de 2Go

Création d'une table

SQL

- Exemple:

```
CREATE TABLE Produit (
  numProd VARCHAR(2)  NOT NULL,
  nomProd VARCHAR(20) NOT NULL,
  couleur VARCHAR(10),
  poids   NUMBER(5,2),
  origine VARCHAR(30)
)
```

indique que ces attributs doivent obligatoirement avoir une valeur

Création d'une table

SQL

- Vous pouvez afficher la description d'une table à l'aide de la commande **DESC**

```
DESC Produit

Nom          Non renseigne NULL? Type
-------------------- -------- ----
numProd                NOT NULL VARCHAR(2)
nomProd                NOT NULL VARCHAR(20)
couleur                         VARCHAR(10)
poids                           NUMBER(5,2)
origine                         VARCHAR(30)
```

Contraintes sur une table

SQL

- **Clé de la relation**

  `CONSTRAINT` *`nomCtr`* `PRIMARY KEY (`*`liste attr`*`)`

- Exemple:

  `CONSTRAINT cleFour PRIMARY KEY (numFour)`

  - Le contrainte **`cleFour`** définit l'attribut **`numFour`** comme étant la clé de la relation **`Fournisseur`**

- Les valeurs d'une clé sont toutes différentes et ne peuvent pas être nulles

Contraintes sur une table

SQL

- **Domaine de validité**

```
CONSTRAINT nomCtr CHECK condition
```

- La syntaxe de la condition booléenne est la même que pour celle de la clause **where**
- Exemple:

```
CONSTRAINT noteOk
    CHECK (note>=0 and note<=20)
```

  - La contrainte **noteOk** définit le domaine de validité de l'attribut **note** de la relation **Suit**

Contraintes sur une table

SQL

- **Contrainte d'intégrité référentielle**

```
CONSTRAINT nomCtr FOREIGN KEY (attr local)
    REFERENCES relRéf(attrRéf)
```

- La valeur de l'attribut local (également appelé clé étrangère) n'est acceptable que si elle appartient à l'ensemble des valeurs de l'attribut de référence
- Exemple:

```
CONSTRAINT fourOk FOREIGN KEY (numFour)
    REFERENCES Fournisseur(numFour)
```

Contraintes sur une table

SQL

- **Attribut à valeur unique**

```
CONSTRAINT nomCtr UNIQUE liste_attributs
```

- Une déclaration de contrainte **unique** est moins forte que **primary key**: dans ce cas, l'unicité est assurée, mais avec possibilité de valeur nulle

Exemples

- Table **Fournisseur**

```
CREATE TABLE Fournisseur (
  numFour VARCHAR(2)  NOT NULL,
  nomFour VARCHAR(15) NOT NULL,
  remise  NUMBER(2),
  ville   VARCHAR(15),
  CONSTRAINT cle_Four PRIMARY KEY (numFour)
)
```

NULL par défaut, i.e. l'attribut peut ne pas avoir été renseigné

Exemples

SQL

- Table **Produit**

```
CREATE TABLE Produit (
  numProd VARCHAR(2)  NOT NULL,
  nomProd VARCHAR(15) NOT NULL,
  couleur VARCHAR(10),
  poids   NUMBER(5,2),
  origine VARCHAR(15),
  CONSTRAINT cle_Prod PRIMARY KEY (numProd)
)
```

Exemples

SQL

- Table **Stock**

```
CREATE TABLE Stock (
  numFour VARCHAR(2)  NOT NULL,
  numProd VARCHAR(2)  NOT NULL,
  qte     NUMBER(4),
  CONSTRAINT cle_Stock
      PRIMARY KEY (numFour,numProd),
  CONSTRAINT fourOk FOREIGN KEY (numFour)
      REFERENCES Fournisseur(numFour),
  CONSTRAINT prodOk FOREIGN KEY (numProd)
      REFERENCES Produit(numProd),
  CONSTRAINT qteOk CHECK (qte>0)
)
```

Suppression d'une table

- Une table est supprimée à l'aide de la commande SQL **DROP TABLE**

```
DROP TABLE Stock
```

- Cette commande supprime non seulement les tuples de la table, mais également la table elle-même
- Il ne sera plus possible d'insérer de tuple dans cette table (puisqu'elle n'existe plus !)

Remarques

- La création d'une clé entraîne également la création d'un **index primaire**
- Ces index permettent d'améliorer les performances lors des interrogations…
- Mais ils ralentissent (très peu) l'insertion d'un tuple dans une table (« insertion triée »)
- On peut créer des index supplémentaires (cf. commande **`CREATE INDEX`** d'Oracle)

Mise à jour

SQL

- L'insertion d'un tuple dans une table est réalisée via la commande **INSERT INTO**

```
INSERT INTO Stock VALUES('f1','p1',300)

INSERT INTO Fournisseur
    VALUES('f1','Dupont',0,'Paris')
```

- Si certains attributs sont déclarés **not null** (cas des clés par ex.), vous **devez** leur donner une valeur
- Sinon, vous pouvez donner la valeur **NULL** (attribut non renseigné)

Mise à jour

SQL

- La destruction de tuples se fait via la commande **DELETE FROM** qui supprime tous les tuples vérifiant une certaines propriété

```
DELETE FROM Stock WHERE(numFour='f1')
DELETE FROM Fournisseur WHERE (remise>10)
```

- La condition booléenne de la clause **WHERE** est soumise aux mêmes règles que celle de l'instruction **SELECT**

Mise à jour

SQL

- La modification des valeurs d'un (ou plusieurs) tuple(s) se fait via la commande **UPDATE**

```
UPDATE Fournisseur
  SET remise=remise+5
  WHERE EXISTS(
    SELECT qte
    FROM Stock
    WHERE Fournisseur.NumFour=Stock.NumFour
      and Qte>500
  )
```

Mise à jour

SQL

- Autre exemple:

```
UPDATE Suit SET note=17.5
  WHERE numEtud in (SELECT numEtud
                    FROM Etudiant
                    WHERE Nom='Pourcel')
    and numCours in (SELECT numCours
                     FROM Cours
                     WHERE nomCours='Java')
```

Mise à jour

- Commandes SQL pour mettre à jour la liste des tuples d'une table:
  - insertion : **`INSERT INTO … VALUES(…)`**
  - suppression : **`DELETE FROM … WHERE …`**
  - modification : **`UPDATE … SET … WHERE …`**

Mise à jour

SQL

- On peut modifier la structure d'une table à l'aide de la commande **ALTER TABLE**
  - pour ajouter un nouvel attribut

```
ALTER TABLE Stock
  ADD montant NUMBER(8,2)
```

  - pour modifier la déclaration d'un attribut

```
ALTER TABLE Produit
  MODIFY COLUMN nomProd VARCHAR(25)
```

Interrogation

- Construction de base d'une requête SQL

```
SELECT a1,…,ap
FROM T1,…,Tn
WHERE B
```

- Sémantique en algèbre relationnelle

$$(T_1 \times \ldots \times T_n)\{B\}[a_1,\ldots,a_p]$$

produit cartésien entre les relations

sélection

projection

Interrogation

- Opérations ensemblistes:

```
<sélection 1> UNION     <sélection 2>
<sélection 1> INTERSECT <sélection 2>
<sélection 1> MINUS     <sélection 2>
```

- Exemple:
  - fournisseurs qui n'ont livré aucun produit
    - `Fournisseur[numFour] - Stock[numFour]`

```
SELECT numFour FROM Fournisseur
MINUS
SELECT numFour FROM Stock
```

Interrogation

- Remarques sur les opérations ensemblistes
  - les deux sélections sont des requêtes **SELECT** dont le nombre et le type des attributs sélectionnés doivent être identiques
  - **ALL**, cité après **UNION**, évite l'élimination des tuples en double
  - dans certaines implémentation de SQL, il se peut que **MINUS** soit noté **EXCEPT**

Interrogation

SQL

- Le **renommage** permet de définir des alias sur les noms des tables
  - quand une même table apparaît plusieurs fois dans une jointure, des sous-requêtes,…
  - quand il y a ambiguïté sur le nom d'un attribut (même attribut dans plusieurs tables)
  - pour simplifier l'écriture des requêtes
- Exemple:

```
select C1.nomCours, C2.nomCours
from Cours C1, Cours C2
where C1.nomProf=C2.nomProf
```

Interrogation

SQL

- Opérateurs utilisables dans les prédicats:
  - opérateurs de comparaison `>`, `<`, `>=`, `<=`, `=`, `<>` (Oracle admet aussi `^=` et `!=` pour ce dernier)
  - opérateurs logiques **`and`**, **`or`**, **`not`**
  - prédicats sur les valeurs des attributs
    - **`between`**, **`not between`**
    - **`null`**, **`not null`**

Interrogation

SQL

- Prédicats (suite):
  - fonctions sur les chaînes de caractères
    - concaténation (`||`)
    - sous-chaîne (`substr`)
    - opérateur `like` avec les caractères
      - `%` pour une chaîne quelconque de 0 à n caractères
      - `_` pour un caractère et un seul
  - fonctions sur les ensembles
    - `in`, `not in`
    - `exists`, `not exists`
    - `some`, `any`, `all`

Interrogation

SQL

- Prédicats (exemple):
  - nom des cours suivis par au moins un étudiant

```
select nomCours from Cours C
where exists(select * from Suit S
                where C.numCours=S.numCours)
```

  - nom du (des) fournisseur(s) consentant la plus forte remise

```
select nomFour from Fournisseur
where remise >= all(select remise
                    from Fournisseur)
```

Interrogation

SQL

- Pour faire des calculs, tant dans la partie **select** que dans les prédicats
  - opérateurs arithmétiques **+**, **-**, *****, **/**
  - fonctions statistiques applicables sur des groupes de tuples
    - fonctions numériques: **avg**, **sum**, **min**, **max**
    - nombre d'éléments: **count**

Interrogation

SQL

- Exemples:
  - totaux des heures de cours, de TD et de TP pour un prof donné

```
select sum(nbHC), sum(nbHTD), sum(nbHTP)
from Cours
where nomProf = 'Munier'
```

  - nombre de prof différents faisant des cours

```
select count(distinct nomProf)
from Cours
where nbHC > 0
```

Interrogation

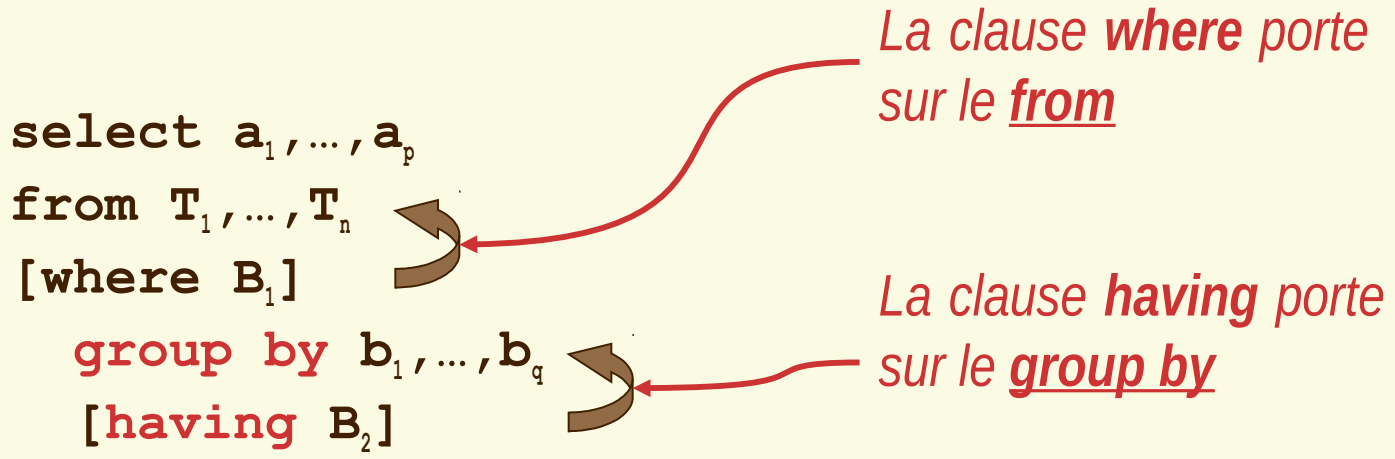
SQL

- Exemples (suite):
  - nombre d'étudiants ayant eu en Java une note supérieure à la moyenne du module Java

```
select count(*) from Suit
where numCours = 'C1'
  and note >= (select avg(note) from Suit
               where numCours = 'C1')
```

Interrogation

- Il est possible de faire des calculs pour un ensemble de tuples vérifiant un même critère: requêtes avec **partitionnement**



Interrogation

SQL

- La clause **`group by`** permet de réaliser des agrégats
- Un agrégat est un partitionnement horizontal d'une table en sous-tables en fonction des valeurs d'un ou plusieurs attributs de partitionnement

Interrogation

SQL

- Exemple:

```
select nomCours, count(*) "nb etud"
from Cours,Suit
where Cours.numCours=Suit.numCours
group by nomCours
```

Affiche, pour chaque nom de cours, le nombre d'étudiants qui suivent ce cours

Interrogation

SQL

- Exemple:

```
select numFour, count(numProd), sum(qte)
from Stock
group by numFour
```

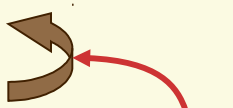
Donne, grâce à un regroupement des tuples par fournisseur, pour chaque fournisseur

- le nombre de produits livrés
- la somme des quantités livrées (tous produits confondus)

Interrogation

- Exemple:

```
select nomFour, count(numProd), sum(qte)
from Stock
group by numFour
  having count(*)>=3
```



s'applique à cette sous-table

Idem, mais uniquement pour les fournisseurs ayant livré au moins 3 produits

Interrogation

SQL

- Exemple:

```
select nomFour
from Fournisseur
group by nomFour
  having count(*)>=2
```

Délivre, du fait de la condition imposée par la clause **having** sur les groupes sélectionnés, les noms de fournisseurs présents au moins deux fois dans la table (cette requête détecte les homonymes)

Interrogation

SQL

- Edition des résultats:
  - on veut éditer la liste des produits livrés par les fournisseurs et les quantités livrées, triées selon le nom des fournisseurs et le nom des produits avec affichage du cumul des quantités (tous produits confondus) par fournisseur

```
select nomFour,nomProd,qte
from Fournisseur F,Stock S,Produit P
where F.numFour=S.numFour
  and P.numProd=S.numProd
```

Interrogation

SQL

- Edition des résultats:
  - **rupture et tri**: pour organiser cet état en groupes, chaque groupe correspondant à un fournisseur, on utilise l'ordre **break** suivant:

```
rem definit une rupture sur le nom de four.
break on nomFour
```

  - Ce qui nécessite de mettre un **order by** sur le **select**:

```
order by nomFour,nomProd
```

Interrogation

SQL

- Edition des résultats:
  - **calcul des sous-totaux**: pour obtenir pour chaque groupe, i.e. pour chaque fournisseur, le cumul des quantités, on utilise l'ordre **compute** suivant:

```
rem a chaque rupture sur le nom de four,
rem un cumul des quantites sera fourni
compute sum of qte on nomFour
```

Interrogation

SQL

- Edition des résultats:
  - **fignolage de la présentation**: on peut utiliser divers ordres pour améliorer la présentation de l'état:
    - prévoir un titre de haut et de bas de page (ordres `ttitle` et `btitle`)
    - renommer les colonnes de la requête (ordre `column`)
    - sauter une ligne après chaque groupe (clause `skip`)

Interrogation

SQL

```
break on nomFour skip 1
rem rupture sur nomFour avec saut d'une ligne

column nomProd heading 'nom du|produit'
rem la barre verticale fait que 'produit' est
rem mis sous 'nom du'

ttitle 'liste des fournisseurs avec leurs produits'
rem ttitle=top title (haut de page)

btitle 'rapport mensuel'
rem btitle=bottom title (bas de page)

compute sum of qte on nomFour
rem cumul des quantites à chaque changement du nom
```

Interrogation

SQL

```
rem la requete elle-meme
select nomFour,nomProd,qte
  from Fournisseur F,Stock S,Produit P
  where F.numFour=S.numFour
    and P.numProd=S.numProd
  order by nomFour,nomProd

rem annulation des ordres de presentation, des
rem ruptures et calculs associes
column nomProd clear
ttitle off
btitle off
clear breaks
clear computes
```

Interrogation

SQL

```
Ve Fev 25                                                          page    1
                    liste des fournisseurs avec leurs produits

                nom du
NOMFOUR         produit                 QTE
--------------- --------------- ----------
Alpages         chemise                 100
                jean                    400
                pantalon                200
                veste                   200
                veste longue            300
**************** ----------
sum                                    1200

Courvite        chemise                 150
                manteau                  50
                pantalon                400
                veste                   300
**************** ----------
sum                                     900

                                  rapport mensuel
Appuyez sur 'Return' pour continuer ...
```

Interrogation

SQL

- Notion de vue:
  - Une **vue** est une « manière de voir » les données figurant dans la base
  - Les vues sont des **relations virtuelles**
    - qui ne contiennent aucune donnée par elles-mêmes
    - que l'on peut manipuler comme des relations réelles
  - Les vues permettent de créer un **sous-modèle** du modèle principal de la BdD

Interrogation

SQL

- Exemple de vue:
  - relation (virtuelle) regroupant les fournisseurs de Paris

```
create view FourParis
as select numFour,nomFour,remise
from Fournisseur
where ville='Paris'
```

Interrogation

SQL

- Exemple de vue:
  - vue sur la table `Stock` restreinte aux fournisseurs parisiens

```
create view StockParis
as select S.numFour,S.numProd,S.qte
from FourParis F,Stock S
where S.numFour=F.numFour
```

Interrogation

SQL

- Exemple de vue:
  - on peut maintenant obtenir le nom des produits livrés par des fournisseurs parisiens

```
select distinct nomProd
from StockParis S,Produit P
where S.numProd=P.numProd
```

Interrogation

SQL

- Utilisation des vues:
  - **Simplifier** l'accès aux données en décomposant un problème en sous-problèmes (un peu comme avec des variables temporaires en programmation)
  - **Simuler** les sous-requêtes sur les bases de données ne supportant pas les requêtes imbriquées
  - Seule la **définition** de la vue est enregistrée dans la base ➔ **table virtuelle**

Interrogation

SQL

- Confidentialité:
  - Nous sommes propriétaire de toute table ou vue que nous créons
  - Par défaut, les données sont privées, donc réservées à leur propriétaire
  - Les privilèges sont:
    - `select`
    - `insert`
    - `update`
    - `delete`
    - `references`

`all` *représente la liste de tous les privilèges*

Interrogation

- Confidentialité:
  - SQL permet
    - d'accorder (**grant**) des privilèges à d'autres utilisateurs sur
      - nos tables et nos vues
      - les tables et les vues pour lesquelles nous avons reçu des privilèges avec transmission possible (**with grant option**)
    - de retirer (**revoke**) des privilèges que nous avons accordés à d'autres, ainsi que les privilèges éventuellement transmis par ceux à qui nous les retirons

Interrogation

SQL

- Confidentialité:
  - `grant` accorder un privilège
  - `on` sur une table ou une vue
  - `to` à un utilisateur, un groupe d'utilisateurs ou à tous (public)
  - `with grant option` transmission possible (facultatif)
- Exemple:
  - autorise `select`,`insert` et `update` sur la table `Produit` pour `gallon`

```
grant select,insert,update
on Produit
to gallon
```

Interrogation

SQL

- Exemples:
  - supprime tous les droits sur la table **Produit** pour **a2g1e4**

```
revoke all on Produit from a2g1e4
```

  - accès en consultation pour tout le monde sur la vue **FourParis** vue précédemment

```
grant select on FourParis to public
```

Interrogation

SQL

- Concurrence d'accès (survol):
  - quand on fait des mises-à-jour (**insert**, **delete**, **update**) sur des relations, elles ne deviennent effectives qu'après avoir fait un **commit**
  - tant que nous n'avons pas fait un **commit**, il est possible de les annuler avec un **rollback**
  - Idée: toute instruction de mise-à-jour pose un verrou sur les tuples concernés; ce verrou est relâché lors du **commit**

Récapitulatif

SQL

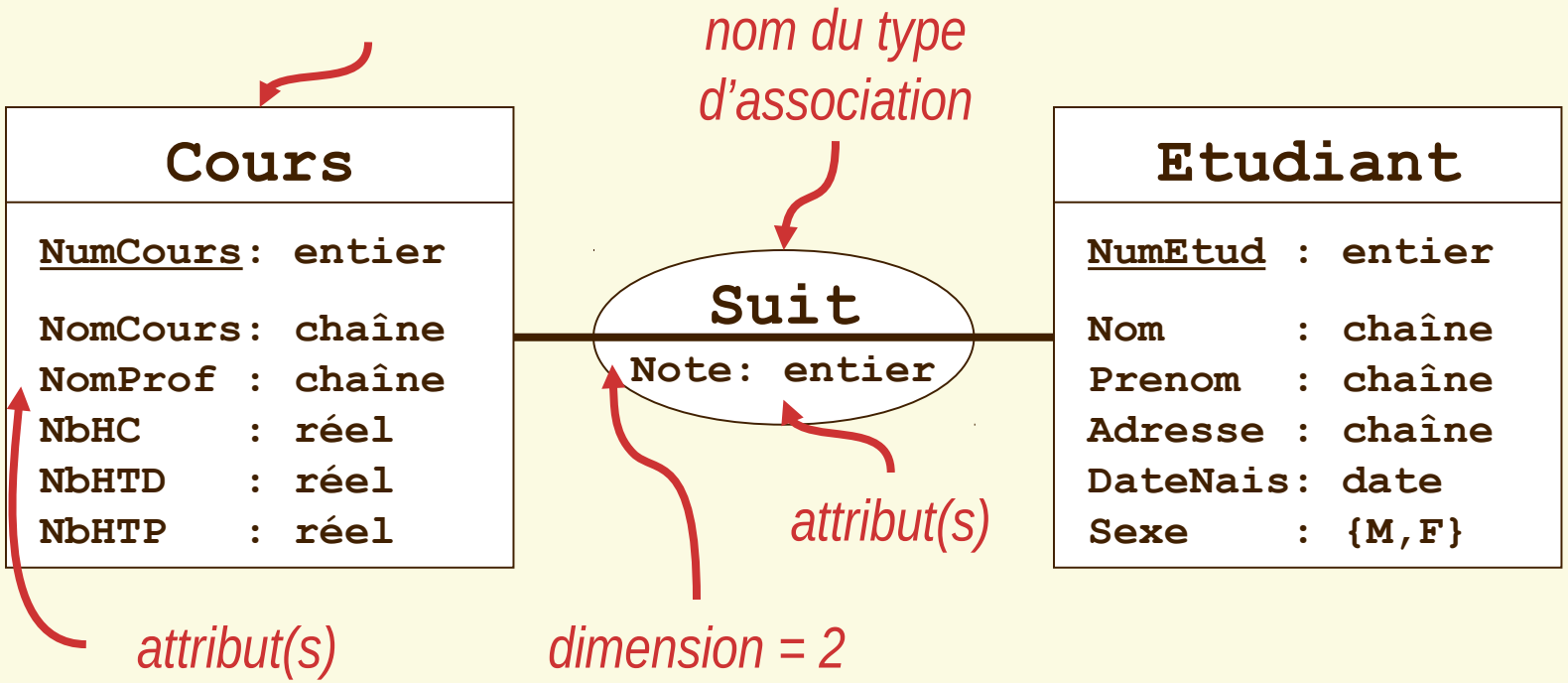
- Définition des données
  - **create table**, **create view**
  - **alter table**
  - **drop table**, **drop view**
- Manipulation des données
  - **select**, **union**, …
  - **insert**, **delete**, **update**
- Exploitation de la base
  - **grant**, **revoke**
  - **commit**, **rollback**, **set transaction**

Plan

- Introduction
- Un survol des bases de données
- Exploitation d'une base de données
  - algèbre relationnelle (un peu…)
  - un langage de requêtes: SQL
- Conception d'une base de données
  - modèle entités-associations
  - modèle relationnel

Conception BdD

- Rappel schéma entités-associations:



Clés

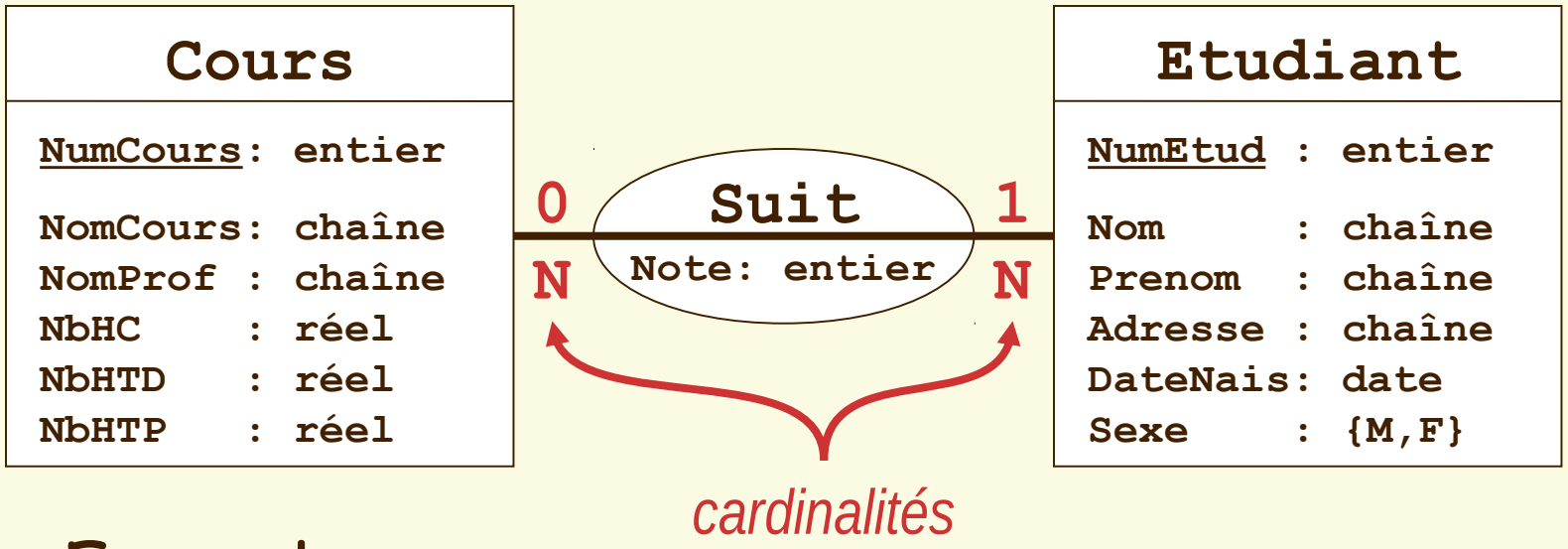
- Les clés des entités doivent être uniques (et avoir une valeur, i.e. `'not null'`)
- Les clés des associations contiennent **au moins** les clés des entités reliées, et doivent elles aussi être uniques (éventuellement en y ajoutant des attributs de l'association)

Cardinalités

- Les **cardinalités** précisent la signification des types d'association
- Les **cardinalités** d'un type d'entité au sein d'un type d'association représentent le **nombre minimum** et le **nombre maximum** d'occurrences d'une entité donnée dans les associations de ce type
- Généralement, on utilise **0**, **1** ou **N** (ou *)

Cardinalités

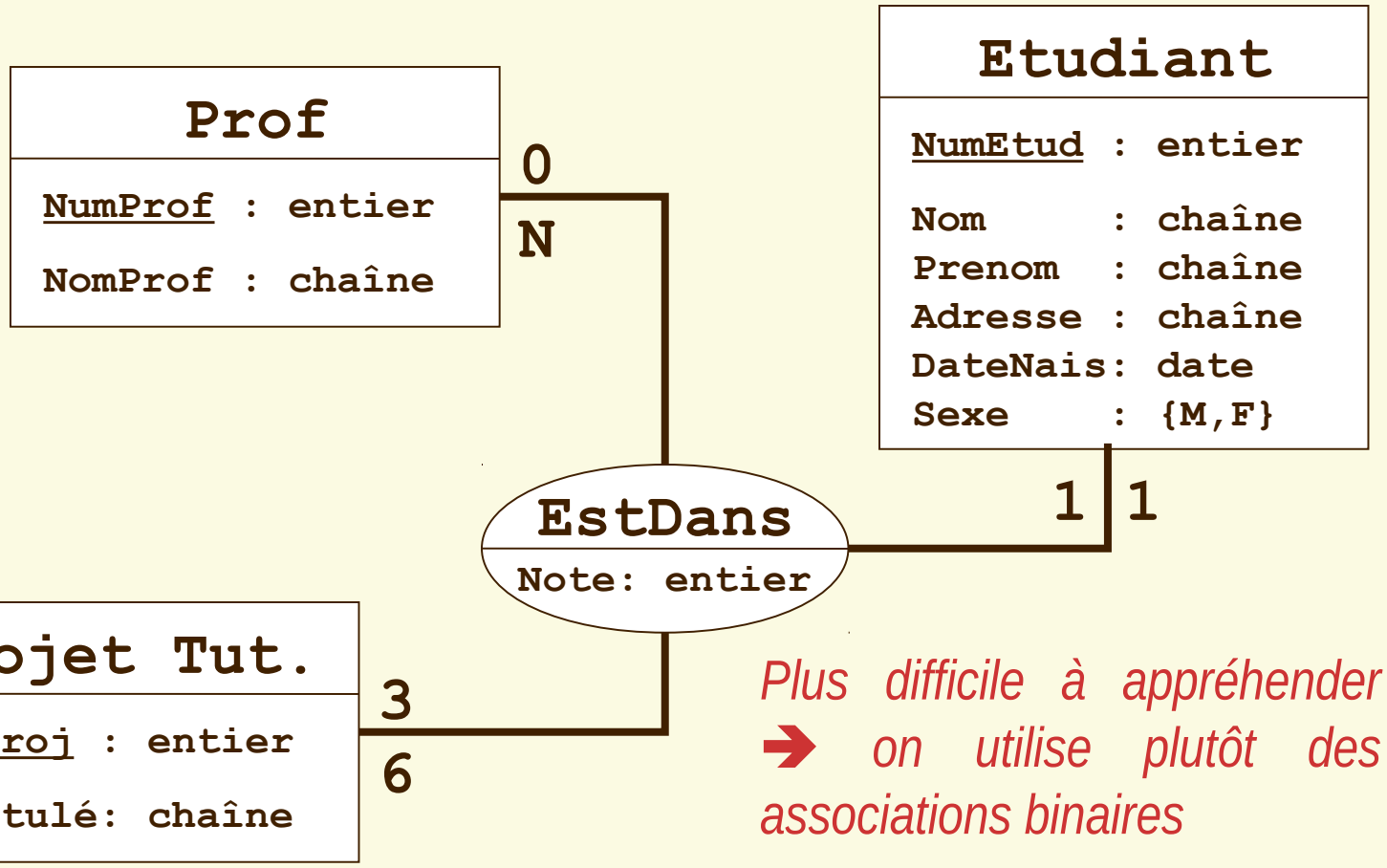
Schéma E/A



- Exemple:
  - nb min de cours suivis par un étudiant : 1
  - nb max de cours suivis par un étudiant : N
  - nb min d'étudiants suivant un cours : 0
  - nb max d'étudiants suivant un cours : N

Cardinalités

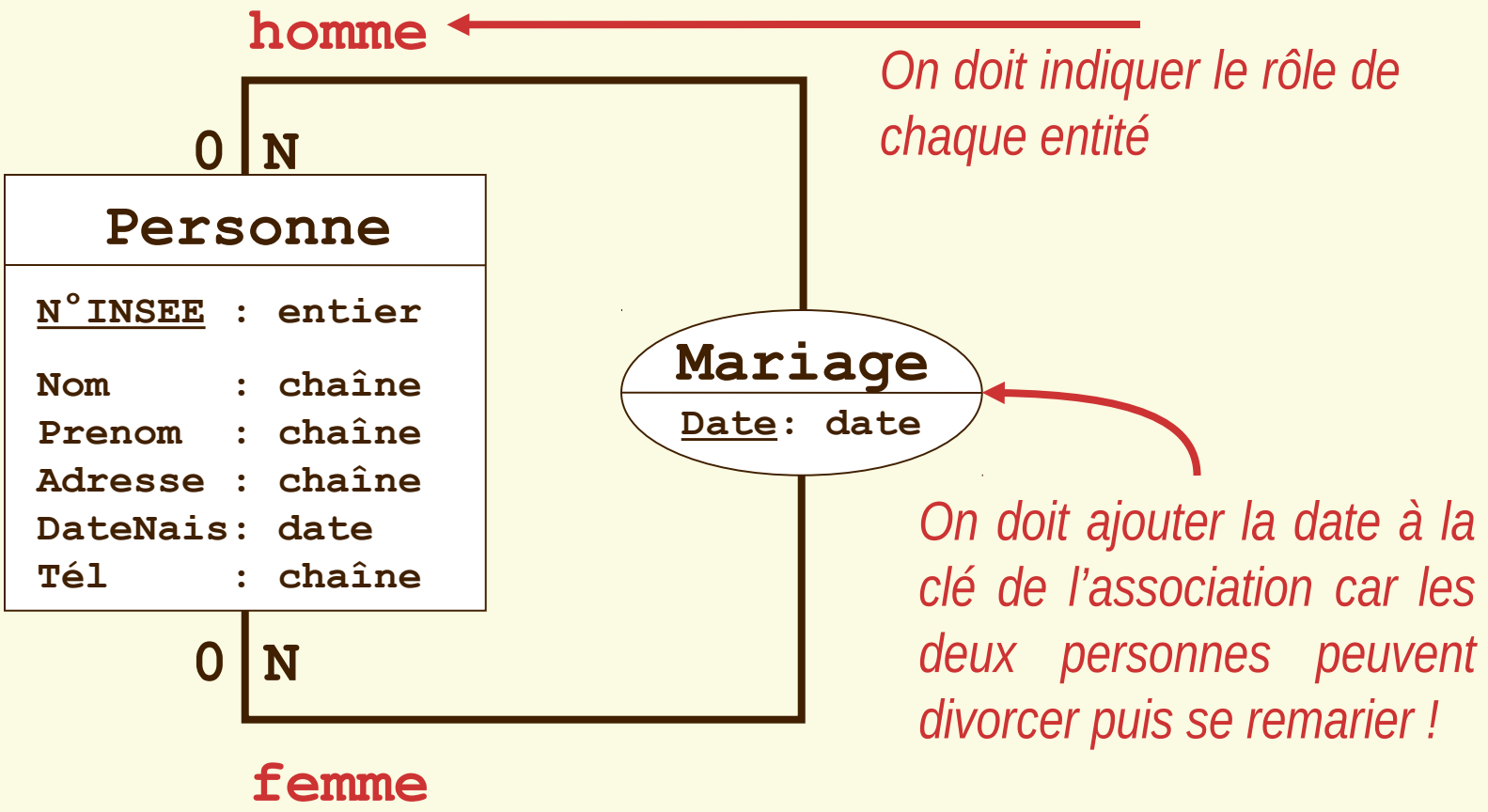
Schéma E/A



Plus difficile à appréhender ➔ on utilise plutôt des associations binaires

Cardinalités

Schéma E/A

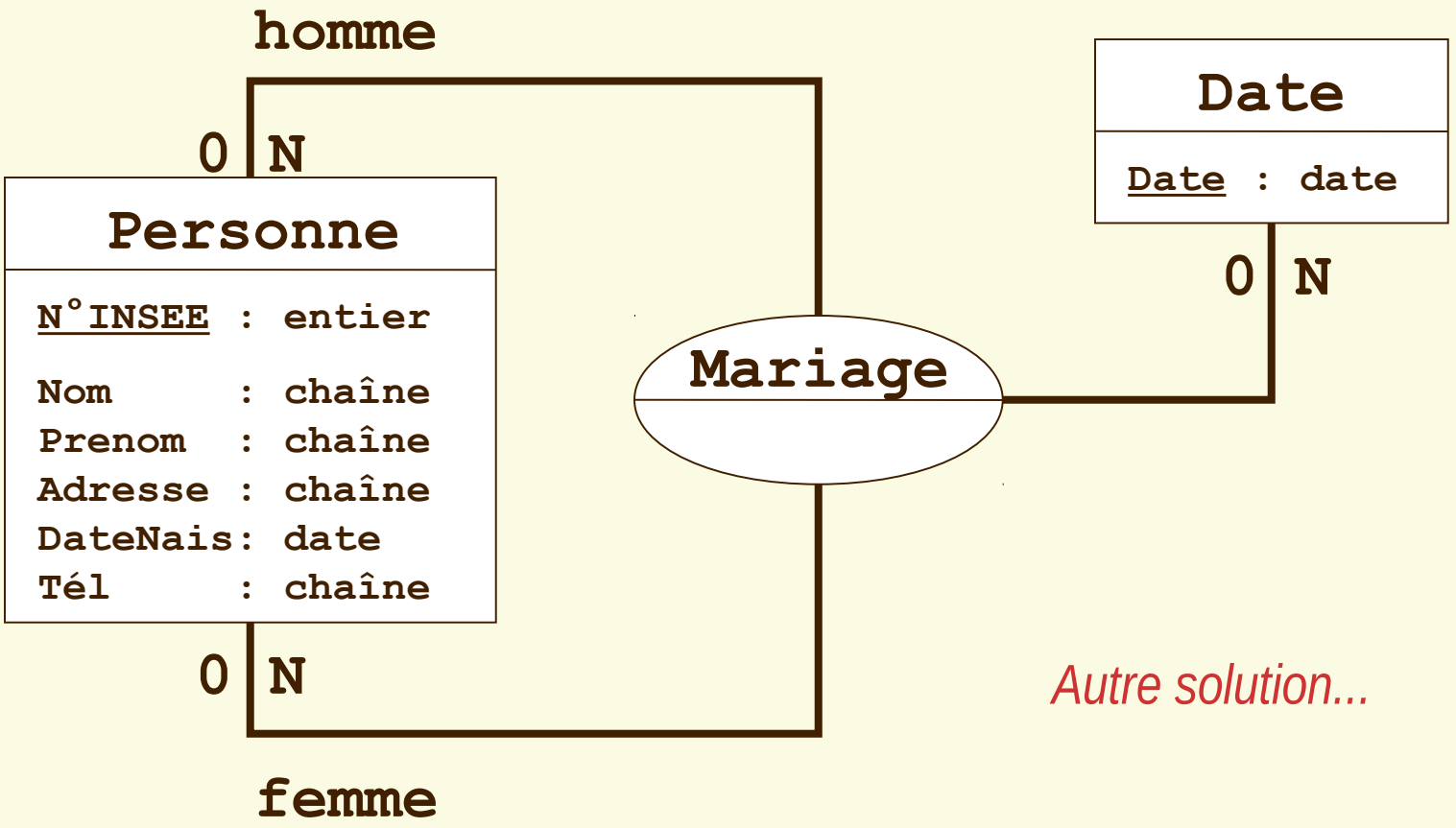


On doit indiquer le rôle de chaque entité

On doit ajouter la date à la clé de l'association car les deux personnes peuvent divorcer puis se remarier !

Cardinalités

Schéma E/A



Autre solution...

Cardinalités

- Remarques:
  - on se limite généralement aux associations **binaires**
  - si un même type d'entité intervient plusieurs fois dans un même type d'association, on doit explicitement indiquer le **rôle** de chaque entité
  - si on a des cardinalités 1,1 de chaque côté, on a une bijection ⇒ il est possible de **fusionner** les deux entités en une seule

Généralisation

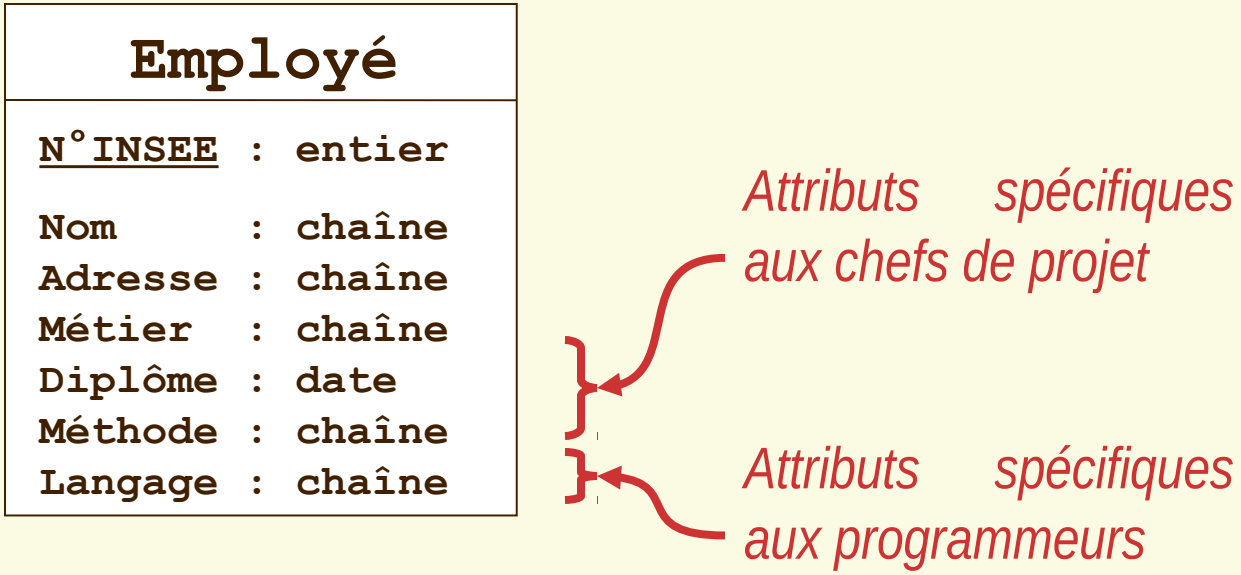
- Exemple:
  - dans une société de services, on a trois types d'employés:
    - les programmeurs
    - les chefs de projet
    - les secrétaires
  - on veut pour chaque catégorie
    - N° INSEE
    - nom
    - adresse

Généralisation

- Exemple (suite):
  - pour les programmeurs, on veut connaître le langage de programmation sur lequel il est spécialisé
  - pour les chefs de projet, on veut connaître leur diplôme le plus élevé ainsi que leur méthode d'analyse

Généralisation

- 1ère solution

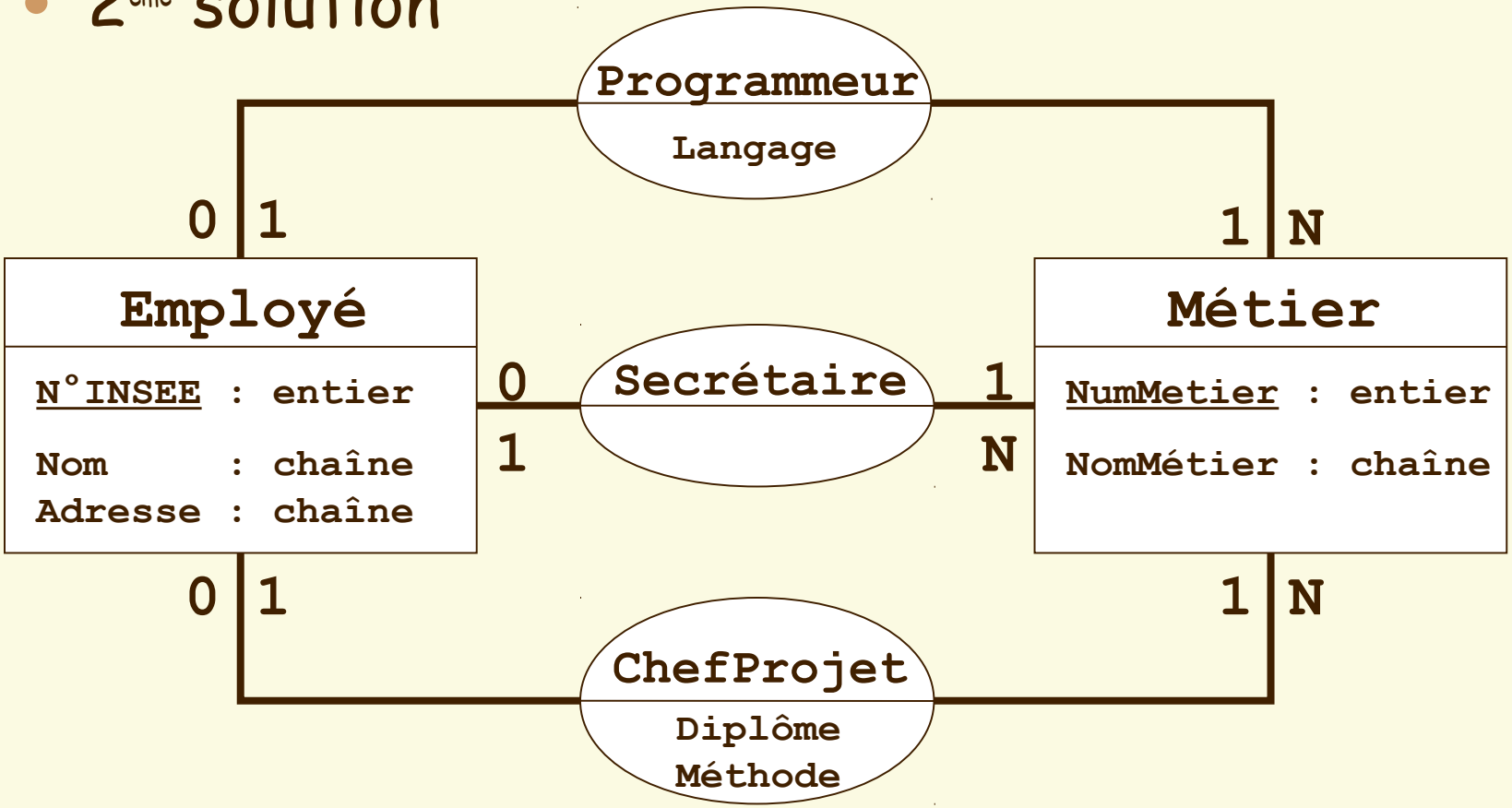


➔ problème des attributs vides

Généralisation

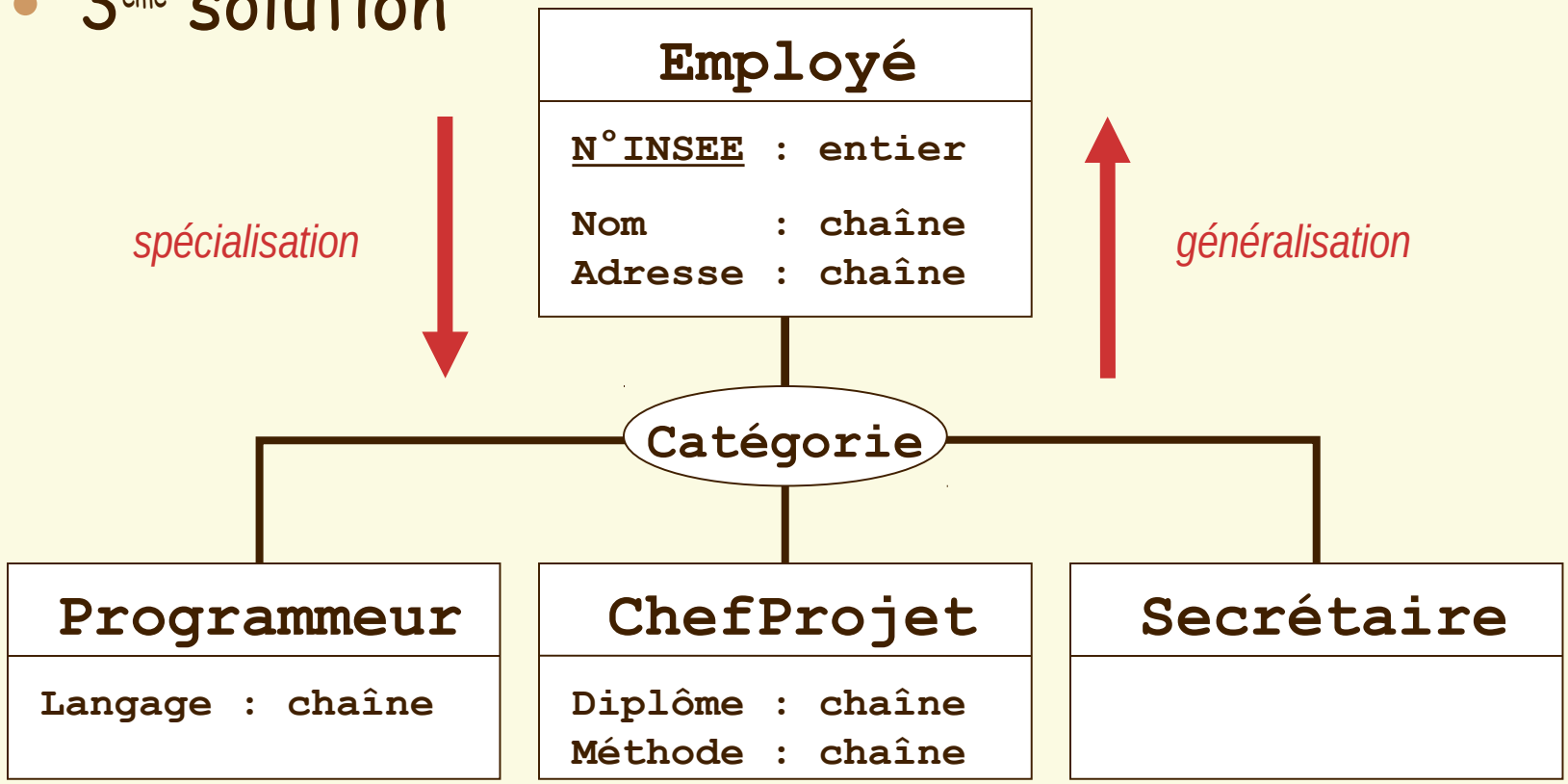
Schéma E/A

- 2ème solution



Généralisation

- 3ème solution



Généralisation

- 3ème solution
  - le concept de programmeur est une **spécialisation** du concept d'employé
  - le concept d'employé est une **généralisation** des trois autres concepts
- La catégorie est le critère de répartition en sous-classes
- C'est l'équivalent de l'héritage en POO

Généralisation

- 3ème solution
  - **Employé(<u>N°INSEE</u>,Nom,Adresse)**
  - **Programmeur(<u>N°INSEE</u>,Langage)**
  - **ChefProjet(<u>N°INSEE</u>,Diplôme,Méthode)**
  - **Secrétaire(<u>N°INSEE</u>)**
- Nous serions arrivés au même résultat si on avait traduit en tables la 2ème solution

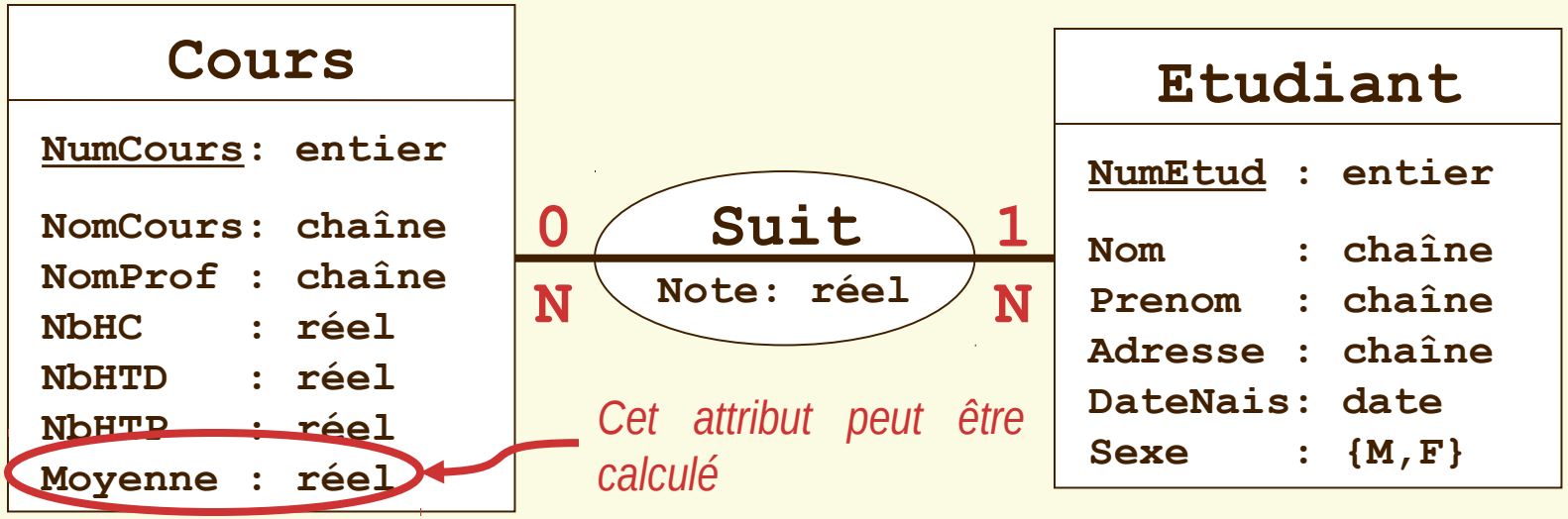
Généralisation

- Elle permet de **regrouper** plusieurs sous-classes dans une super-classe
- En général, les sous-classes forment une **partition** de la super-classe
- Les sous-classes « **héritent** » des attributs et relations de la super-classe
- Il peut y avoir des associations entre les sous-classes
  - Ex: un chef de projet dirige des programmeurs

Limites du modèle E/A

Schéma E/A

- C'est un **modèle de données** et non un modèle de traitements
  - ➔ les attributs **déductibles** ou **calculés** ne sont pas représentés



Limites du modèle E/A

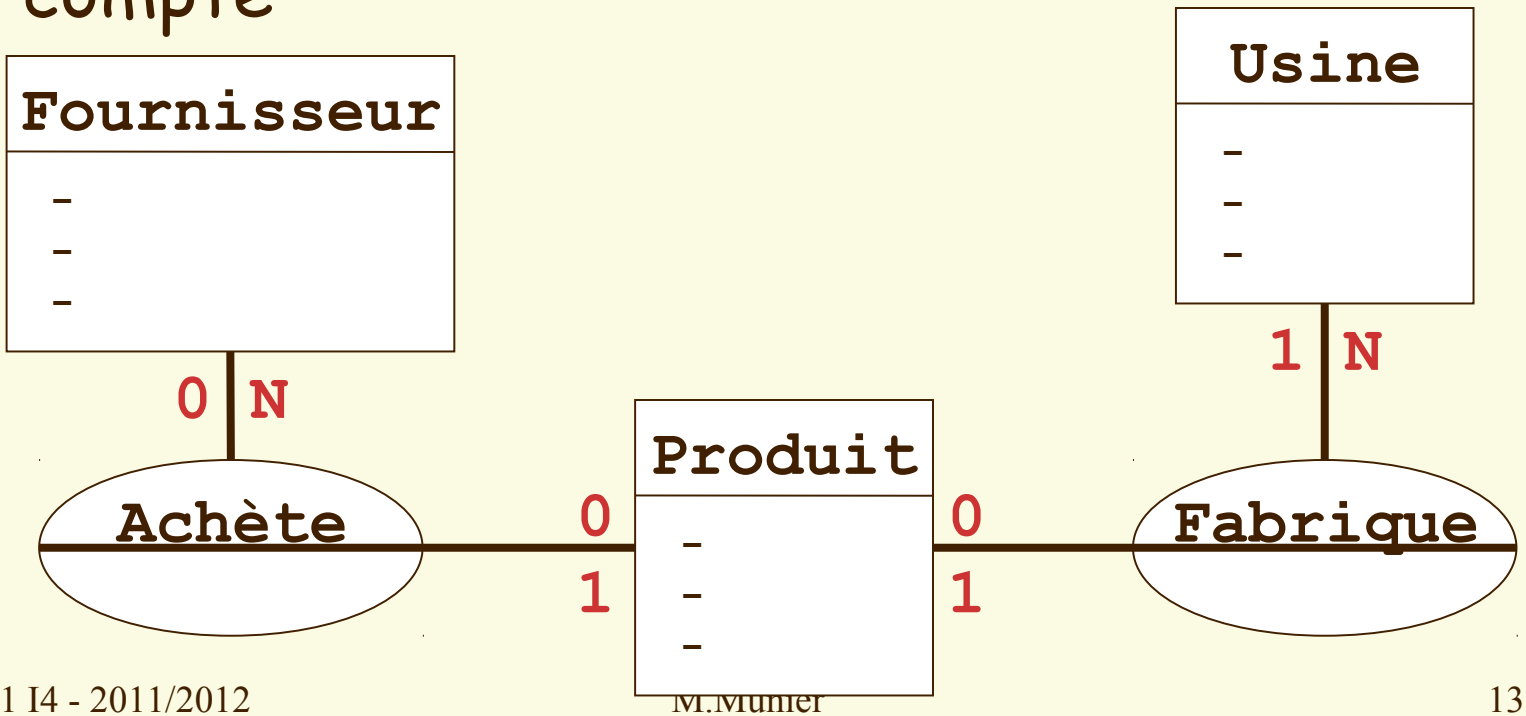
Schéma E/A

- Certaines règles de gestion ne sont pas représentables
  - chaque rentrée scolaire, début septembre, on archive les étudiants de l'année précédente
  - chaque début de semaine on lance un mailing de rappel des contrôles non corrigés (!)
  - on ne peut inscrire de nouveaux étudiants qu'en septembre et octobre

Limites du modèle E/A

Schéma E/A

- La règle « un produit est soit fabriqué, soit commandé » ne peut pas être prise en compte



Limites du modèle E/A

Schéma E/A

- Solution:
  - utiliser un modèle des traitements en complément du modèle de données E/A

  ➔méthodes de conception: Merise, UML,…
- C'est au programme manipulant la BdD de gérer ces règles de fonctionnement
  - certains SGBD offrent:
    - contraintes d'intégrité référentielle (foreign keys)
    - contraintes de validité (check)
    - déclencheurs (triggers)

Plan

- Introduction
- Un survol des bases de données
- Exploitation d'une base de données
  - algèbre relationnelle (un peu…)
  - un langage de requêtes: SQL
- Conception d'une base de données
  - modèle entités-associations
  - modèle relationnel

Dépendance fonctionnelle

- Définition:

Soient **X** et **Y** des attributs (simples ou composés) d'une relation.

> **Y est fonctionnellement dépendant de X** si et seulement si à toute valeur de **X** correspond **au plus** une valeur de **Y**.
>
> On dit aussi que **X détermine Y** et on note **X → Y**

- Corollaire:

> Si 2 n-uplets ont même valeur sur les attributs **X**, alors ils ont même valeur sur ceux de **Y**.

DF élémentaire

- Définition:

On dit que la dépendance fonctionnelle **X** → **Y** est **élémentaire** s'il n'existe pas de **X'** ⊂ **X** tel que **X'** → **Y**.

DF directe

- Définition:

On dit que la dépendance fonctionnelle **X** → **Y** est **directe** s'il n'existe pas de **Z** (**Y** ⊄ **Z** et **Z** ⊄ **X**) tel que **X** → **Z** et **Z** → **Y**.

1NF

- Définition:

Une relation est en **première forme normale** si et seulement si tous ses attributs sont simples, c'est-à-dire s'ils ne sont pas eux-mêmes des relations.

2NF

- Définition:

Une relation est en **deuxième forme normale** si et seulement si:

1. cette relation est en première forme normale
2. toutes les DF issues de la clé sont élémentaires

3NF

- Définition:

Une relation est en **troisième forme normale** si et seulement si:

1. cette relation est en deuxième forme normale
2. toutes les DF sont directes

Plan

- ✓ Introduction
- ✓ Un survol des bases de données
- ✓ Exploitation d'une base de données
  - algèbre relationnelle (un peu…)
  - un langage de requêtes: SQL
- ✓ Conception d'une base de données
  - modèle entités-associations
  - modèle relationnel

Bonus

JDBC

NDLR: Ces informations datent de 2001. Il est donc (fortement) possible que des modifications aient été apportées à cette API.

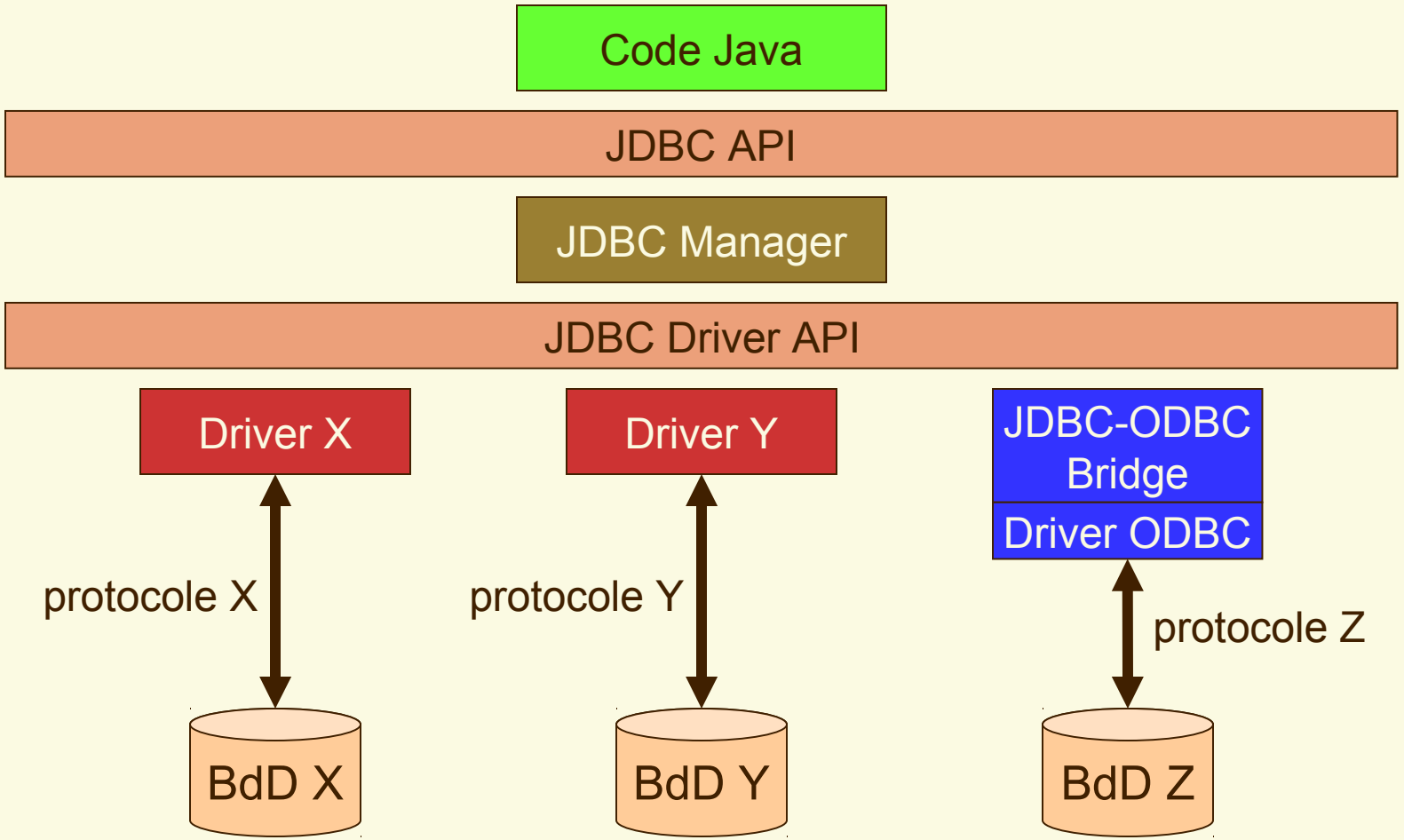
BdD & Java

Bonus

- Idée:
  - Utiliser une BdD SQL directement depuis une application Java
- Solution:
  - JDBC (Java Data Base Connection)
    - Java définit une interface (commune) d'accès à une base de données SQL
    - chaque vendeur de BdD fournit une implémentation (appelée "driver")

BdD & Java

Bonus



BdD & Java

Bonus

- Fonctionnement en 4 étapes
  1. chargement du driver adéquat
  2. connexion du programme à la BdD
  3. préparation puis exécution d'une requête
     - ➔ résultat = liste de n-uplets
  4. boucle d'interprétation du résultat
     - accès séquentiel (ligne par ligne)
     - dans chaque ligne, récupération des attributs

BdD & Java

Bonus

- **java.sql.DriverManager**
  - gère les drivers JDBC
  - on peut indiquer un driver qui existe
    - avec la system property `jdbc.drivers` (ex: Oracle)
    - avec des classes importées (ex: MM pour MySQL)

```
try {
  Class.forName("org.gjt.mm.mysql.Driver");
}
catch (Exception ex)
{… return;}
```

BdD & Java

Bonus

- **java.sql.Connection**
  - établit une connexion avec la BdD (canal TCP)
  - offre des méta-informations sur les tables,...
  - crée les **Statements** (requêtes SQL)
  - gère les transaction (au sens SQL)

```
String url="jdbc:mysql://dax.univ-pau.fr:3306/BDD";
String user="nobody";
String password=null;
Connection myConnection=
    DriverManager.getConnection(url,user,passwd);
```

BdD & Java

Bonus

- **java.sql.Statement**
  - permet d'envoyer une requête SQL vers la BdD
  - crée un **ResultSet** pour stocker le résultat

```
String requete="select NOM,AGE from Etudiant";
Statement st=myConnection.createStatement();
ResultSet rs=st.executeQuery(requete);
```

BdD & Java

Bonus

- **java.sql.ResultSet**
  - le résultat d'une requête est un objet spécial
  - on y accède
    - ligne par ligne (méthode next)
    - avec un positionnement absolu ou relatif
  - on extrait les "colonnes" (méthodes getXXX)

```
while (rs.next()) {
  String lenom = rs.getString("NOM");
  int lage = rs.getInt("AGE");
  System.out.println(lenom + " " + lage);
}
```

BdD & Java

Bonus

```
try {
  Class.forName("org.gjt.mm.mysql.Driver");
} catch (Exception ex) {… return;}

String url="jdbc:mysql://dax.univ-pau.fr:3306/BDD";
String user="nobody";
String password=null;
Connection myConnection=
    DriverManager.getConnection(url,user,passwd);

String requete="select NOM,AGE from Etudiant";
Statement st=myConnection.createStatement();
ResultSet rs=st.executeQuery(requete);

while (rs.next()) {
  String lenom = rs.getString("NOM");
  int lage = rs.getInt("AGE");
  System.out.println(lenom + " " + lage);
}
```

Bonus

NDLR: Ces informations datent de 2001. Il est donc (fortement) possible que des modifications aient été apportées à cette API.

PHP

Bonus

- PHP ≡ "PHP: Hypertext PreProcessor"
- PHP = langage de script embarqué dans les pages HTML et traité par le serveur
- PHP ➔ construction dynamique des pages HTML à partir de résultats (calculs, requêtes SQL adressées à un SGBD,…)
- Nombreuses extensions à PHP
  - génération de PDF, GIF,… à la volée
  - connexion messageries, serveurs LDAP,…

PHP

Bonus

- PHP comparé à:
  - Microsoft ASP
    - PHP contient beaucoup plus de fonctions qu'ASP
    - PHP supporte quasiment tous les standards du Web
    - PHP est extensible
  - Javascript
    - script traité par le serveur et non par le client (browser) ➔ "portabilité"
  - CGI bin (Perl, Python & co)
    - l'apprentissage de PHP nécessite beaucoup moins d'aspirine… ;-)

PHP

Bonus

- Le langage PHP
  - syntaxe proche du C
  - variables (ex: $nom) faiblement typées
  - tableaux associatifs (dictionnaires en Python)
  - fonctions
  - classes (langage "orienté" objet)

PHP

Bonus

- Objectif d'un script PHP
  - générer du HTML sur sa "sortie standard"
- Exemple

```
<HTML>
  <BODY>
    <?php
      echo "Hello World !<P>"
    ?>
  </BODY>
</HTML>
```

PHP

Bonus

- Pourquoi générer des pages HTML dynamiquement avec une BdD ?
  - publications d'articles (ex: slashdot, linuxfr, freshmeat), d'offres d'emploi
  - enregistrement des saisies d'un formulaire
  - vitrine électronique, catalogue sur le Web
- Quelle infrastructure ?
  - trio infernal **PHP** + **MySQL** + **Apache** sur une machine **Linux** (ou FreeBSD)

PHP

Bonus

```
<?php
  $login="mylogin";
  $pass ="mydbpass";
  $db=mysql_connect("localhost",$login,$pass);
  mysql_select_db("mydb",$db);

  $sql="select NOM,AGE from Etudiant";
  $result=MySQL_query($sql,$db);

  while($myrow=MySQL_fetch_array($result))
  {
    $nom=$myrow["NOM"];
    $age=$myrow["AGE"];
    echo "$nom est agé(e) de $age ans";
  }
?>
```