



# Module

# Modélisation UML pour

# les bases de données

# 2018-2019

Michel Dubois

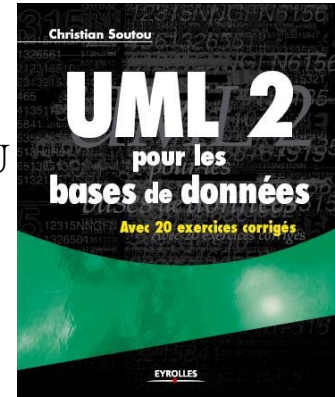
I.U.T. de Vannes

*Michel.Dubois@univ-ubs.fr*

# PLAN (757 transparents)

1.	Rappels sur la démarche UML	<a href="#"><u>006</u></a>
2.	Préalable à la modélisation, le dictionnaire des données	<a href="#"><u>056</u></a>
3.	Rappels sur la modélisation Entité-Association	<a href="#"><u>088</u></a>
4.	Introduction aux notations UML	<a href="#"><u>147</u></a>
5.	Correspondances Merise / UML	<a href="#"><u>191</u></a>
6.	Modèle Logique des Données relationnel	<a href="#"><u>213</u></a>
7.	Autres formalismes pour AGL bases de données	<a href="#"><u>261</u></a>
8.	Rappels sur la modélisation directe	<a href="#"><u>307</u></a>
9.	Modèle Physique Oracle SQL2	<a href="#"><u>370</u></a>
10.	ORM : Correspondance Objet – Relationnel	<a href="#"><u>429</u></a>
11.	Niveau externe avec les vues SQL2	<a href="#"><u>537</u></a>
12.	Les problèmes liés au contexte multi-utilisateurs (droits et concurrence)	<a href="#"><u>581</u></a>
13.	Les passerelles JDBC	<a href="#"><u>681</u></a>

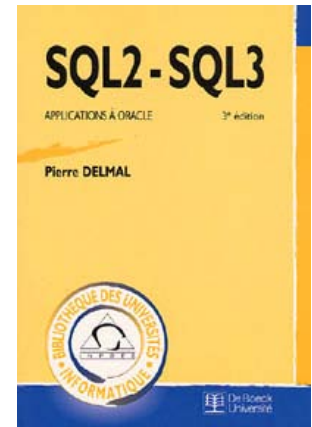
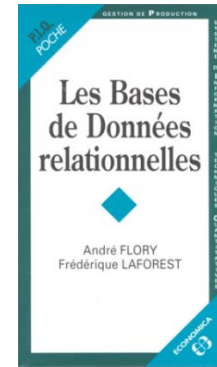
# Bibliographie (1/3)



- UML et bases de données
  - «*UML 2 pour les bases de données avec 20 exercices corrigés*», C. Soutou, Eyrolles, 2007. cote BU: 005.74 SOU et 005.111 UML s et 2<sup>ième</sup> BU: 005.13 SOU
  - 2<sup>ième</sup> édition Mai 2012 (30 exercices, les aspects Objet-relationnels atténués)
  - 3<sup>ième</sup> édition Mai 2015, cote BU: 005.74 SOU, autres notations ERD
  - «*Conception de bases de données avec UML*», G. Roy, Presses de l'Université du Quebec, 2007. cote 005.74 ROY
- Pour récupérer le standard UML
  - [www.omg.org](http://www.omg.org) et [www.celigent.com/uml](http://www.celigent.com/uml)
- Pour comprendre le paradigme objet
  - «*Object-Oriented Software Construction - 2nd edition*», B. Meyer, Prentice Hall, 1997. côte BU: VALORIA 00210. Traduction Eyrolles 2008 côte BU: 005.1 MEY
- Pour s'initier au langage UML
  - «*Modélisation Objet avec UML*», P-A. Muller, deuxième édition, Eyrolles, 1999. côte BU: 005.111 UML m et 878 UML MUL
  - «*The Unified Modeling Language User Guide*», Grady Booch, Jim Rumbaugh, Ivar Jacobson, Addison-Wesley, 1999. côte BU: 878 UML BOO et 878 BOO  
Traduction française Eyrolles 2000. côte BU: 878 UML BOO
  - «*Modélisation et conception orientées objet avec UML2*», de Michael Blaha et James Rumbaugh, 2<sup>ème</sup> édition, Pearson Education France, 2005 – Traduction de l'ouvrage *Applying Object-Oriented Modeling and Design with UML*, Prentice Hall 2005. Cote de la BU: 005.11 BLA

# Bibliographie (2/3)

- Pour la modélisation conceptuelle
  - « *Analysis Patterns: Reusable Objects Models* », M. FOWLER, Addison-Wesley, 1997. cote BU: 878 FOW
  - « *Advanced Object-Oriented Analysis & Design Using UML* », James J. Odell, SIGS Reference Library, Cambridge University Press, 1998. cote BU: 878 ODE
- Pour une introduction aux bases de données relationnelles
  - « *Les bases de Données Relationnelles* », A. FLORY, F. LAFOREST, Economica. 1996 1<sup>ère</sup> édition cote BU: AR 872 FLO  
2002 2<sup>ième</sup> édition cote BU: 004 FLO  
2005 3<sup>ième</sup> édition cote BU: 872 FLO
  - « *SQL2-SQL3 – Applications à Oracle* », Pierre Delmal, 3<sup>ième</sup> édition, 2001, cote 874 SQL DEL



Pour le développement de bases de données (pas à la BU)

- “*Ingénierie des Systèmes d’Information MERISE*”, D. Nanci et al. SYBEX.
- « *The Data Modeling Handbook : A Best-Practice Approach to Building Quality Data Models* », Michael Reingruber, William W. Gregory, Paperback.
- « *Object-Oriented Modeling and Design for Database Applications* », M. Blaha et W. Premierlani, Prentice Hall.



# Bibliographie (3/3)

- Pour la conception d'une base de données objet-relationnelle
  - « *de UML à SQL – conception de bases de données* », C. SOUTOU, Eyrolles, 2002. côte BU: 005.74 SOU
  - « *Objet-relationnel sous Oracle8 – Modélisation avec UML* », C. SOUTOU, Eyrolles, 1999. cote BU: 872 SOU
- Pour une étude récente des bases de données objet-relationnelles
  - « *Bases de données orientée-objet – concepts, mise en œuvre et exercices résolus* », C. CHRISMENT, G. CABANAC, K. PINEL-SAUVAGNAT, O. TESTE, M. TUFFERY, Hermes-Lavoisier, 2011. cote BU: 005.74 CHR
- Pour les aspects techniques spécifiques à Oracle 8, 8i, 9i, 10g, 11g
  - « *Programmer Objet avec Oracle – Concepts et pratiques* », C. SOUTOU. Vuibert, 2004. cote BU: 005.74 ORA s pour Oracle 9i et 10g
  - « *Programmer Objet avec Oracle – Concepts et pratiques – 2<sup>ième</sup> édition* », C. SOUTOU. Vuibert, 2008. cote BU: 005.74 ORA s pour Oracle 9i, 10g et 11g
  - « *Objet-relationnel sous Oracle8 – Modélisation avec UML* », C. SOUTOU, Eyrolles, 1999. cote BU: 872 SOU pour Oracle 8
  - « *SQL2-SQL3 – Applications à Oracle* », Pierre Delmal, 3<sup>ième</sup> édition, 2001, cote 874 SQL DEL , chapitres 10, 12 pour Oracle 8i





# Université de Bretagne Sud

*Informatique*



## Génie Logiciel avec UML



### Chapitre 1

## Rappels sur la démarche UML

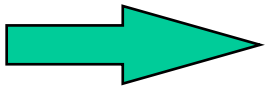
[Retour au plan](#)

## *Objectifs du chapitre*

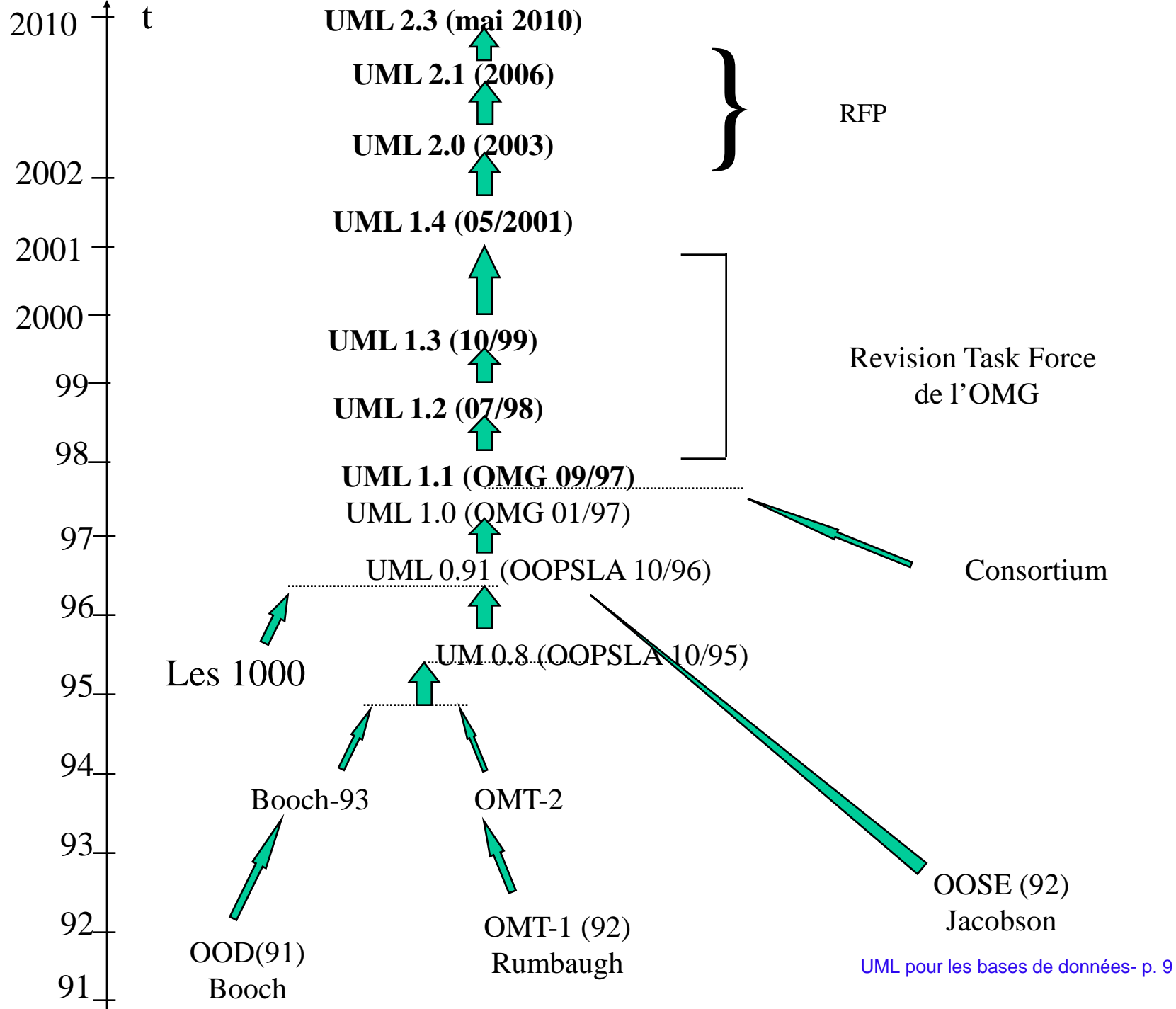
- *Connaître l'origine d'UML et son objectif*
- *Connaître le contenu d'UML*

# 1. Origine

- ❑ 3 ans d'efforts de nombreux méthodologistes du DOO dont 3 réputés au sein de *Rational Software Corporation*
  - *G. Booch* (OOD) fin 94: très expressive en phases de conception et d'implémentation
  - *J. Rumbaugh* (OMT) fin 94 : très utile pour l'analyse et la mise en place des SI
  - *I. Jacobson* (OOSE) fin 95 : excellent support des cas d'utilisation pour l'expression des besoins, l'analyse et la conception haut niveau
- ❑ Le support de grands acteurs du monde de l'informatique fin 96 : IBM, MICROSOFT, HP, ORACLE, DEC, UNISYS, etc.



*Standard de modélisation des applications informatiques construites à l'aide d'objets, depuis Septembre 97 par l'OMG (V 1.1.)*



## ☐ UML 1.2 (la forme uniquement sur réclamations (issues))

- ☐ Corriger des erreurs typographiques et grammaticales
- ☐ supprimer certaines incohérences
- ☐ clarifier certains termes vagues ou ambigus
- ☐ améliorer la lisibilité et la structure du document

## ☐ UML 1.3 (le (petit) fond)

- ☐ relation entre cas d 'utilisations
- ☐ amélioration des diagrammes d 'activités
- ☐ etc.

## ☐ UML 1.4 (la forme)

## ☐ UML 2.0 (le grand fond)

- ☐ formalisation des *profile*
- ☐ meilleure gestion du développement orienté composant
- ☐ amélioration du noyau du méta-modèle UML
- ☐ gestion de la complexité pour les machines à états
- ☐ extension et amélioration d 'OCL ...

# OMG UML Contributors

Aonix  
Colorado State University  
Computer Associates  
Concept Five  
Data Access  
EDS  
Enea Data  
Hewlett-Packard  
IBM  
I-Logix  
InLine Software  
Intellicorp  
Kabira Technologies  
Klasse Objecten  
Lockheed Martin

Microsoft  
ObjecTime  
Oracle  
Ptech  
OAO Technology Solutions  
Rational Software  
Reich  
SAP  
Softeam  
Sterling Software  
Sun  
Taskon  
Telelogic  
Unisys  
...

## *2. Le document UML 1.3 (1034 pages!)*

- ❑ Chap1 : UML summary
- ❑ Chap2 : UML Semantics
- ❑ Chap3 : UML Notation Guide
- ❑ Chap4 : UML Extensions
- ❑ Chap5 : UML Corba Facility Interface
- ❑ Chap6 : UML XMI DTD Specification
- ❑ Chap7 : Object Constraint Language
- ❑ Glossary
- ❑ Appendix
- ❑ Index



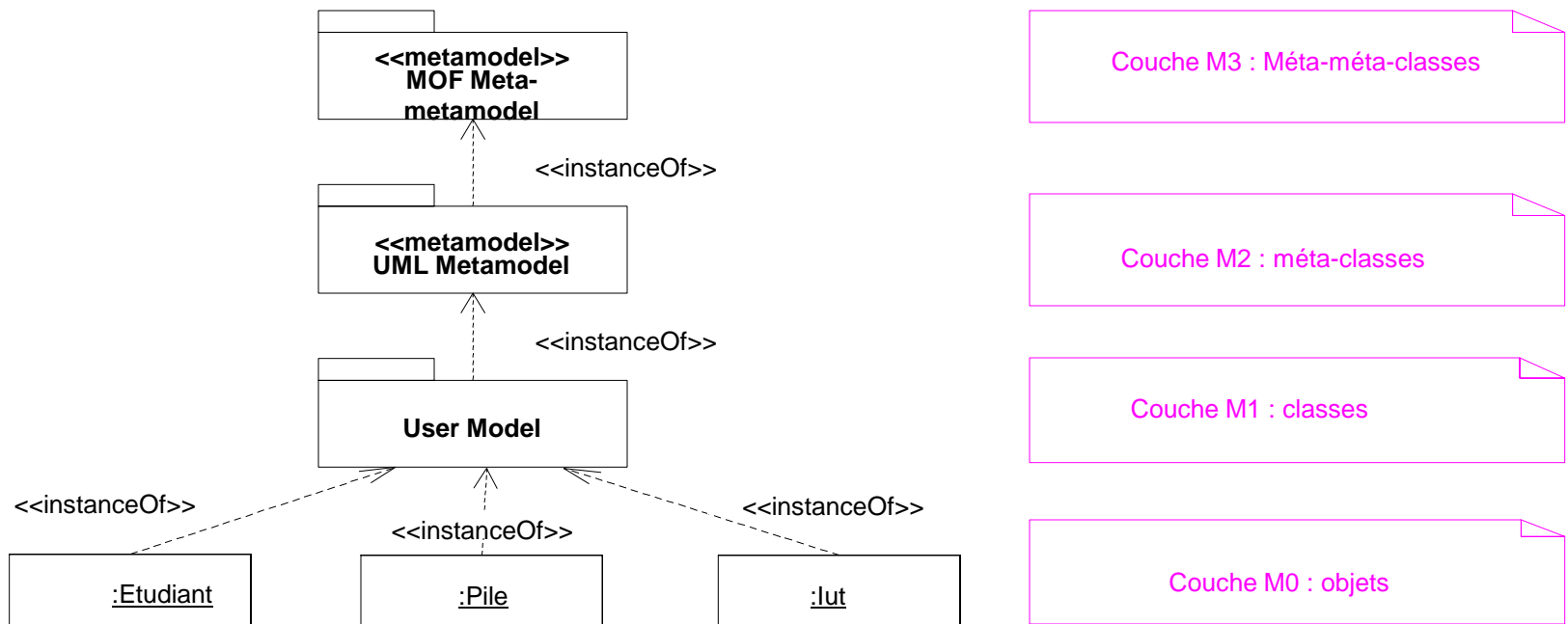
# langage = syntaxe + sémantique

- syntaxe = éléments du langage (les mots) sont assemblés pour former des expressions (des phrases)
- sémantique = la signification des expressions syntaxiques
  - *UML Notation Guide* – définit la syntaxe graphique d'UML
  - *UML Semantics* – définit la sémantique des constructions syntaxiques

- Example of semantic rule: Class [1]
  - **English:** If a Class is concrete, all the Operations of the Class should have a realizing Method in the full descriptor.
  - **OCL:** **not** self.isAbstract **implies** self.allOperations->forAll (op | self.allMethods->exists (m | m.specification-> includes(op)))

- Example of syntactic rules: Class
  - ***Basic Notation:*** A class is drawn as a solid-outline rectangle with three compartments separated by horizontal lines.
  - ***Presentation Option:*** Either or both of the attribute and operation compartments may be suppressed.
- Example of syntactic guideline: Class
  - ***Style Guideline:*** Begin class names with an uppercase letter.

# OMG Metamodel Architecture



# MDA : Model Driven Architecture

- MDA est une proposition de l'OMG dont l'objectif est la conception de systèmes basée sur la seule modélisation du domaine, en faisant abstraction des aspects technologiques. A partir de cette modélisation, MDA propose d'obtenir par transformation les éléments techniques capables de fonctionner au sein d'une plateforme logicielle comme java ou .net.
- Dans MDA, le modèle des objets du domaine s'appelle PIM (Platform Independent Model). Ce modèle est ensuite transformé en un modèle spécifique à une plateforme et à un langage de programmation : PSM (Platform Specific Model).
- UML est un très bon candidat comme langage au niveau du PIM. UML possède l'avantage de décrire finement des objets tout en restant indépendant des technologies. Au niveau des traitements, l'extension Action Semantics d'UML doit lui permettre de répondre à tous les besoins de description.
- [DB-MAIN](#) est un outil CASE qui permet de considérer comme PIM le modèle d'objet d'UML ou le modèle Entité-Association étendu à l'héritage. DB-MAIN offre un passage immédiat entre ces deux modèles. Le PSM est le modèle relationnel.

# 3. Les briques de base d'UML

- Le vocabulaire d'UML inclut 3 types de briques :
  - Les éléments : abstractions, éléments primordiaux d'un modèle
  - Les relations : ce qui lie les éléments entre eux
  - Les diagrammes : ce qui regroupe les collections intéressantes d'éléments

# Les éléments

- Les éléments peuvent être de 4 types :
  - Les éléments structurelles :
    - Classe : collection d'objets qui partagent les mêmes attributs, opérations, relations et sémantique
    - Interface : collection d'opérations qui spécifie un service d'une classe ou d'un composant : elle décrit le comportement externe d'un élément
    - Collaboration : société d'éléments qui travaillent ensemble pour fournir un comportement coopératif qui soit plus que la somme des parties
    - Cas d'utilisation : description d'un ensemble de séquences d'actions qu'effectue un système dans le but de fournir un résultat intéressant pour un acteur donné
    - Classe active, composant, nœud : semblables aux classes mais suffisamment différents pour être traités séparément et nécessaires pour modéliser certains aspects d'un système OO
  - Les éléments comportementales : interaction, machine d'état
  - Les éléments de regroupement : paquetages
  - Les éléments d'annotation : note

# Les relations


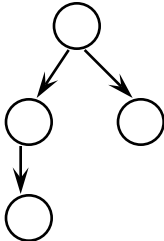
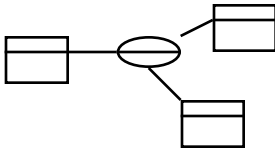

- Les relations peuvent être de 4 types
  - Dépendance : relation sémantique entre 2 éléments dans laquelle un changement dans l'élément indépendant peut affecter la sémantique de l'élément dépendant
  - Association : relation structurelle qui décrit un ensemble de liens, un lien étant une connexion entre objets (agrégation : association entre un tout et ses parties)
  - Généralisation : relation de généralisation/spécialisation dans laquelle les objets de l'élément spécialisé sont substituables aux objets de l'élément générique
  - Réalisation



# Les diagrammes

- C'est la représentation graphique d'un ensemble d'éléments connectés par un ensemble de relations
- Un diagramme est une projection du système selon un point de vue spécifique.
- En UML, 9 diagrammes différents sont gérés
  - *Diagramme de classes*
  - *Diagramme d'objets*
  - Diagramme de séquence
  - Diagramme de collaboration
  - *Diagramme de cas d'utilisation*
  - Diagramme d'états-transitions
  - Diagramme d'activités
  - Diagramme de composants
  - Diagramme de déploiement

### 3. Contenu

Une approche	Une démarche	Un Langage	Des guides (savoir-faire)
			
AOO unifiée	En cours de définition (Objectory Process). Processus d'OMT, d'OOSE	9 types de diagrammes avec possibilité d'extensions par stéréotypage	Ouvrages de Fowler, Jacobson, Booch, etc.

The Unified Modeling Language (UML) is a *language* for Specifying, visualizing, constructing and documenting the *artifacts of software systems*, as well as for *business modeling* and other *non-software systems*. The UML represents a *collection of the best engineering practices* that have proven successful in the modeling of *large and complex systems*.

*OMG Unified Modeling Language Specification*  
*1.3. March 2000*

# UML : un langage de spécification

- Spécifier = construire des modèles précis, non ambigus et complets
- UML aborde la spécification de toutes les décisions importantes d 'analyse, de conception et d 'implémentation qui doivent être faites lors du développement et du déploiement d 'un système logiciel

# UML : un langage pour visualiser

- Communication facilitée : si un développeur écrit directement sous forme de code les modèles qu 'il a en tête, cette information sera perdue ou seulement partiellement récupérable à partir de l 'implémentation quand il s 'en ira
- Langage graphique : certains éléments concernant un logiciel ne peuvent être compréhensibles que si l 'on construit des modèles qui transcendent le LP
- A chaque symbole sa sémantique : si chacun adopte sa propre notation, il risque d 'y avoir une divergence forte entre les diverses interprétations des modèles

# UML : un langage de documentation

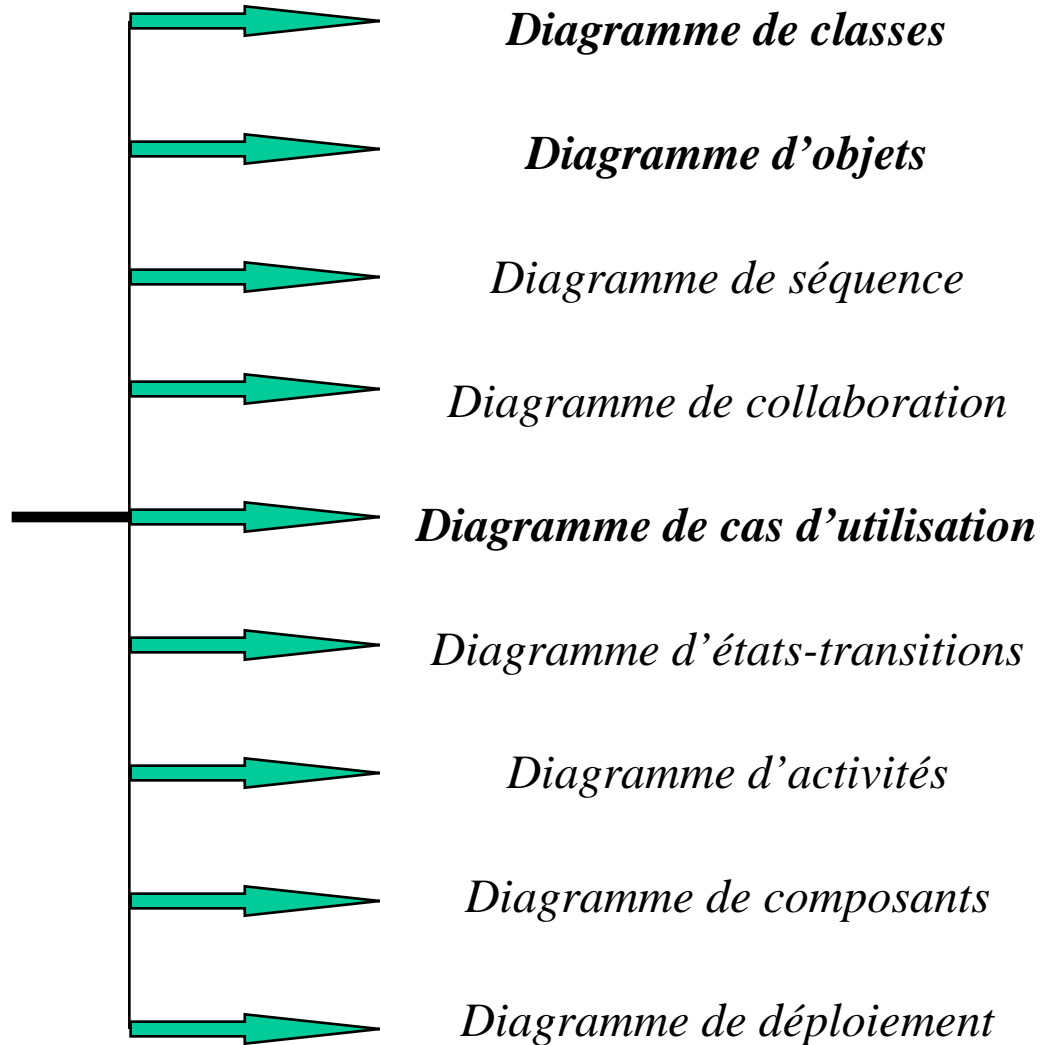
- En plus du code exécutable, une bonne organisation logicielle doit produire les documents présentant
  - L'expression des besoins
  - L'architecture
  - La conception
  - Le code source
  - La planification
  - Les tests
  - Les prototypes
  - Les versions

# UML : un langage d 'implémentation

- Interprétation directe des modèles
  - dans différents LPOO (Java, C++, VB)
  - sous forme de tables de BDR
  - sous forme de stockage persistant d 'une BDO
- Possibilité de génération automatique de code
- Possibilité de faire du *reverse Engineering*



Notation UML





# Les Diagrammes

- **Diagramme de classes** : structure statique du système en termes de classe et de relation entre ces classes au niveau conceptuel, d'analyse ou de conception
- **Diagramme d'objets**
  - Faciliter la compréhension des structures de données complexes décrites dans le DC
  - Identifier les classes et les relations
- **Diagramme de séquence** : montrer les interactions entre objets d'un point de vue temporel sous la forme d'un protocole d'échange de message.

# Les Diagrammes

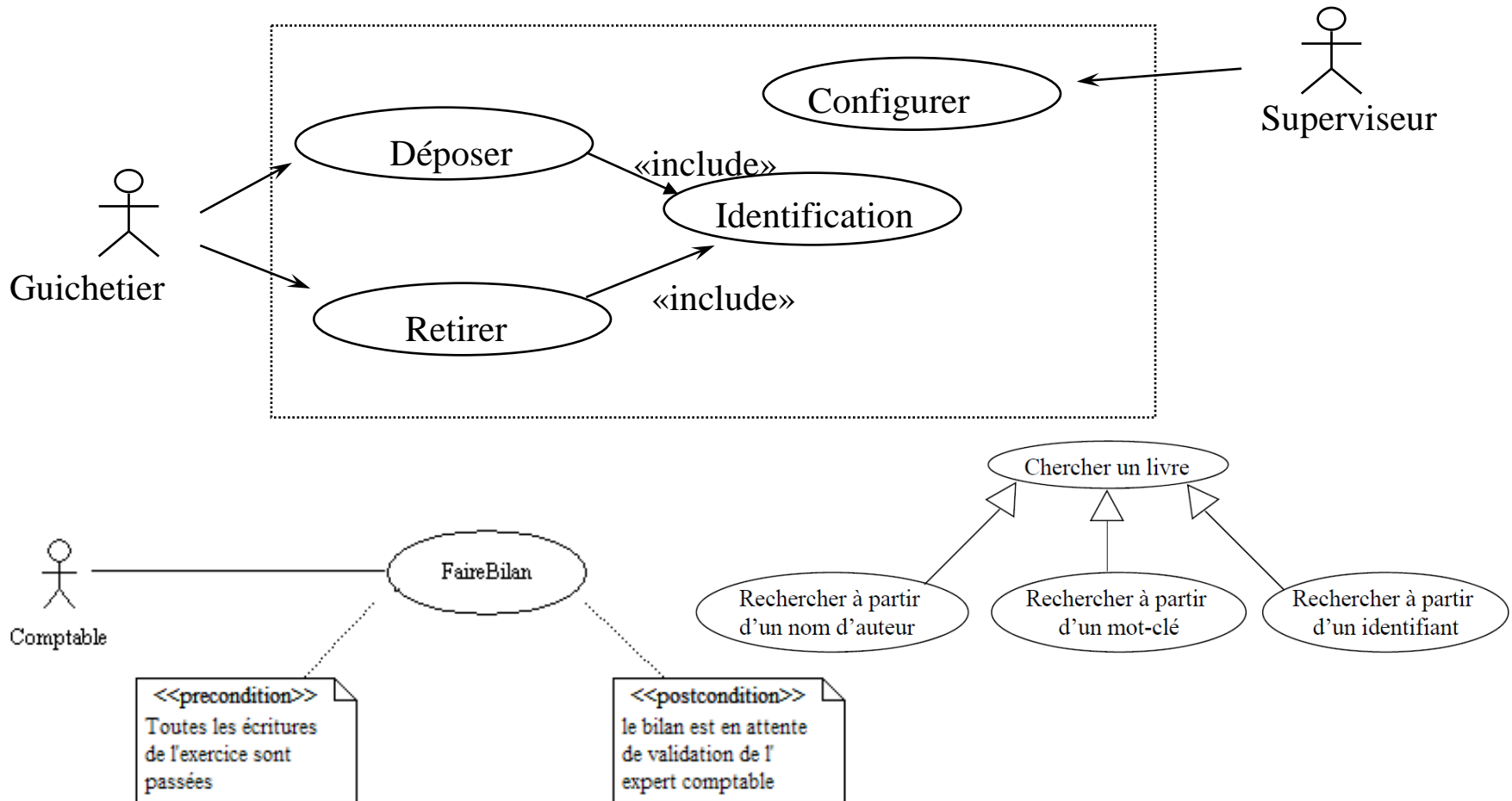
- **Diagramme de collaboration** : montrer les interactions entre les objets en insistant plus particulièrement sur la structure spatiale des objets coopérant
- **Diagramme de cas d'utilisation** : décrire le comportement du système à concevoir du point de vue de l'utilisateur (spécification orientée utilisateur). Pilote le développement.
- **Diagramme d'états-transitions** : modéliser le comportement d'une classe qui a un comportement réactif très marqué suite aux différentes sollicitations externes

# Les Diagrammes

- **Diagramme d'activités** : représenter le comportement interne d'une méthode (la réalisation d'une opération) ou d'un cas d'utilisation ou d'un processus métier
- **Diagramme de composants** : décrire les composants de l'application et leurs relations. Ce diagramme montre les choix de réalisation.
- **Diagrammes de déploiement** : disposition physique des matériels entrant dans la composition du système et la répartition des programmes sur ces matériels.

## 4. Aperçu des notations

### Le diagramme de cas d'utilisation



## *Forme textuelle structurée conceptuelle*

<b>Cas d'utilisation :</b>	Retirer de l'argent
<b>Acteurs :</b>	Guichetier (initiateur)
<b>Type :</b>	Primaire et conceptuel
<b>Description :</b>	Un client arrive au guichet et demande à réaliser un retrait sur l'un de ses comptes. Le guichetier saisit le N° du compte et le montant du retrait. Si le compte est suffisamment approvisionné le logiciel enregistre le retrait et en informe le guichetier.

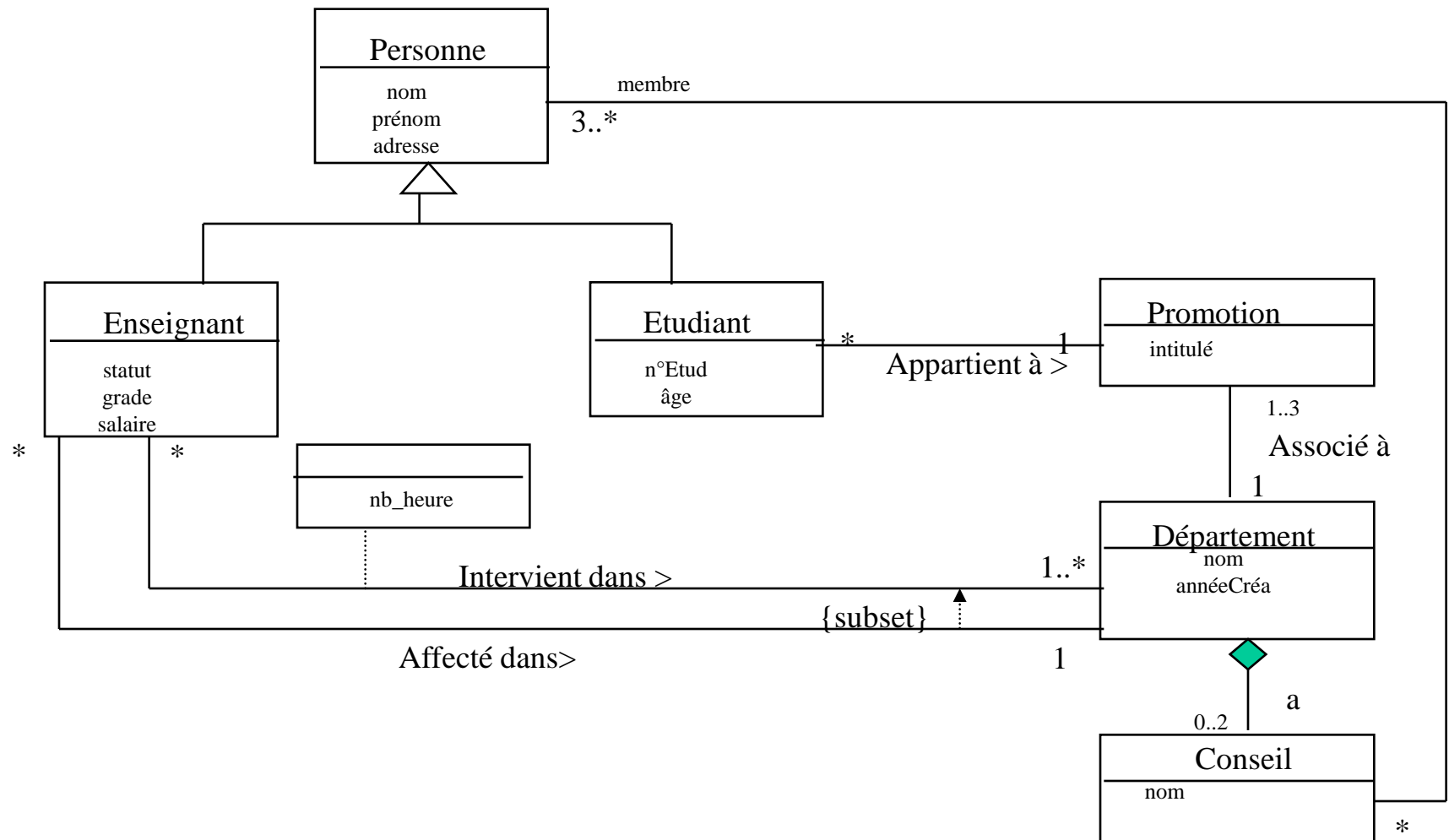
### **Guichetier**

- ☐ 1. Le cas débute lorsque le guichetier sélectionne l'opération de retrait.
- ☐ 3. Le guichetier saisit le N° du compte du client
- ☐ 5. Le guichetier confirme
- ☐ 7. Le guichetier saisit un montant
- ☐ 9. Le guichetier confirme

### **Logiciel**

- ☐ 2. Le logiciel demande le N° du compte concerné
- ☐ 4. Le logiciel vérifie ce N°, affiche son propriétaire et demande confirmation
- ☐ 6. Le logiciel demande le montant du retrait.
- ☐ 8. Le logiciel vérifie que le compte est suffisamment approvisionné et demande confirmation
- ☐ 10. Le logiciel retire le montant du compte
- ☐ 11. Le logiciel notifie la fin

# Le diagramme de classes

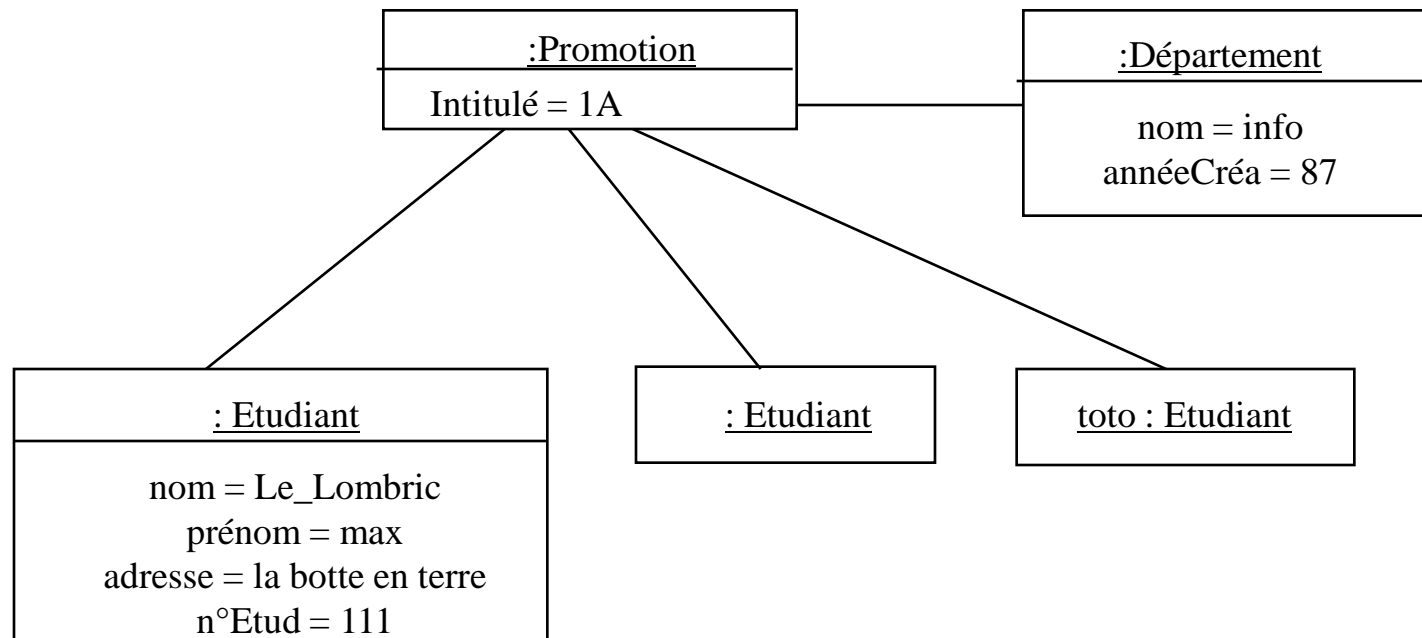


**context Etudiant inv :**

self.âge >= 16

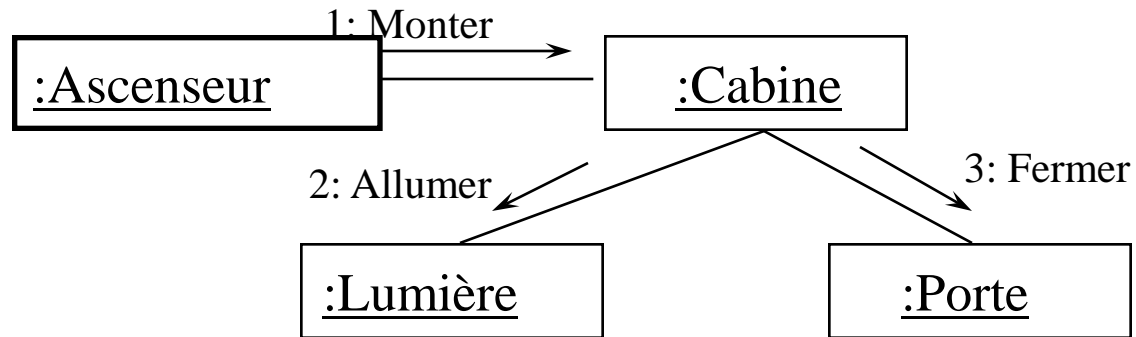
Etudiant.allInstances->forAll{p1,p2| p1<>p2 implies p1.n°Etud <> p2.n°Etud}

# Le diagramme d'objets

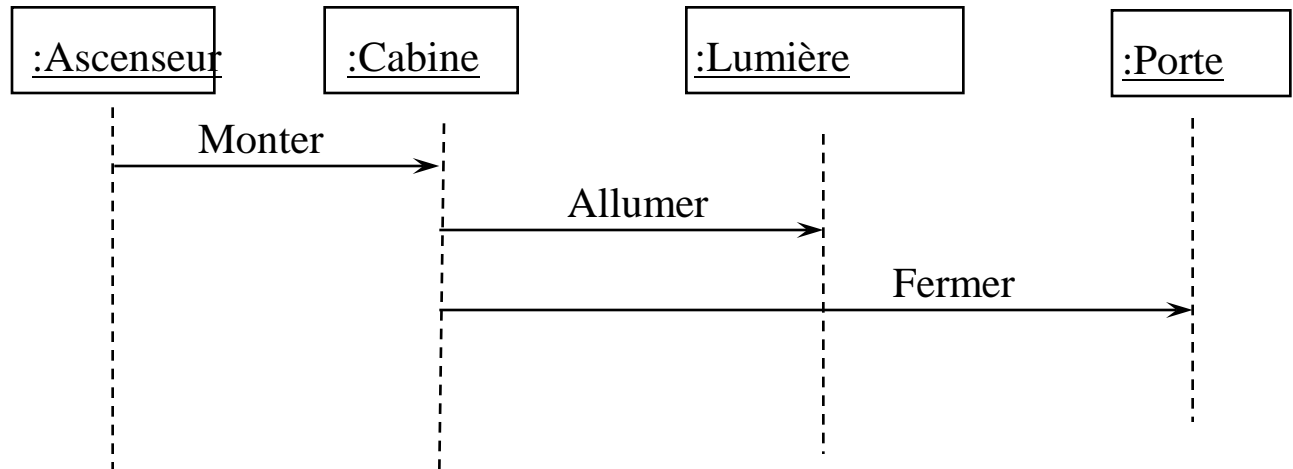


# Les diagrammes d'interaction

## Diagramme de Collaboration

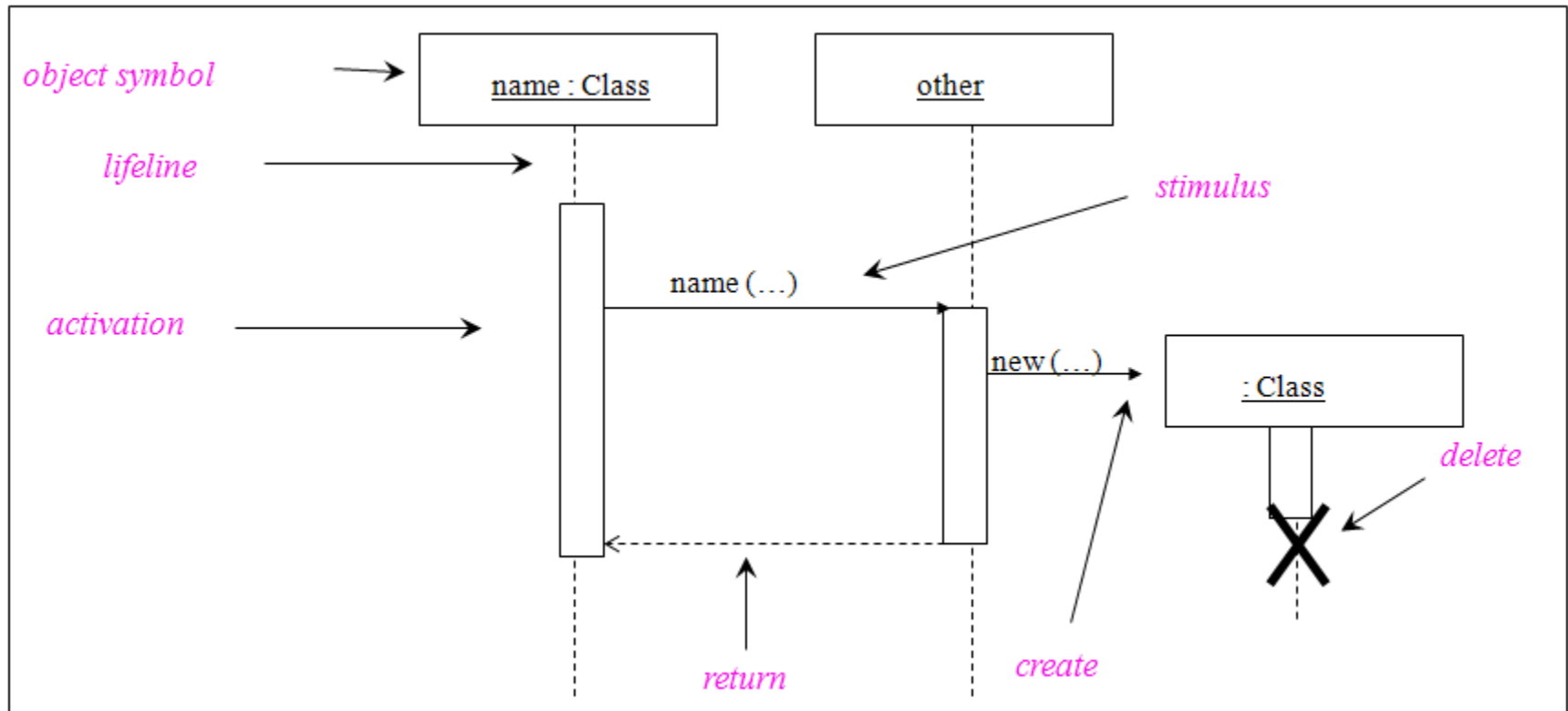


## Diagramme de séquences

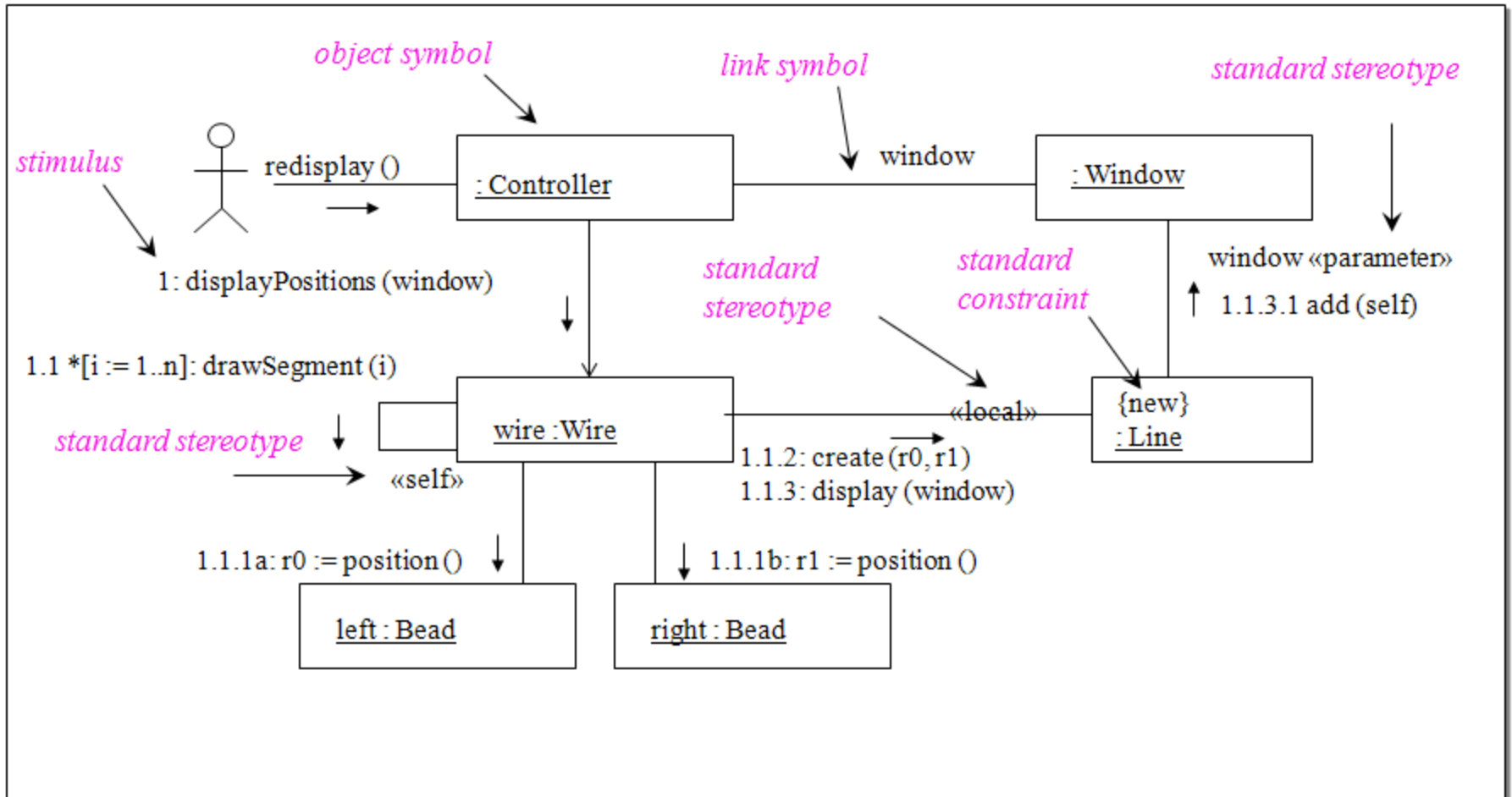




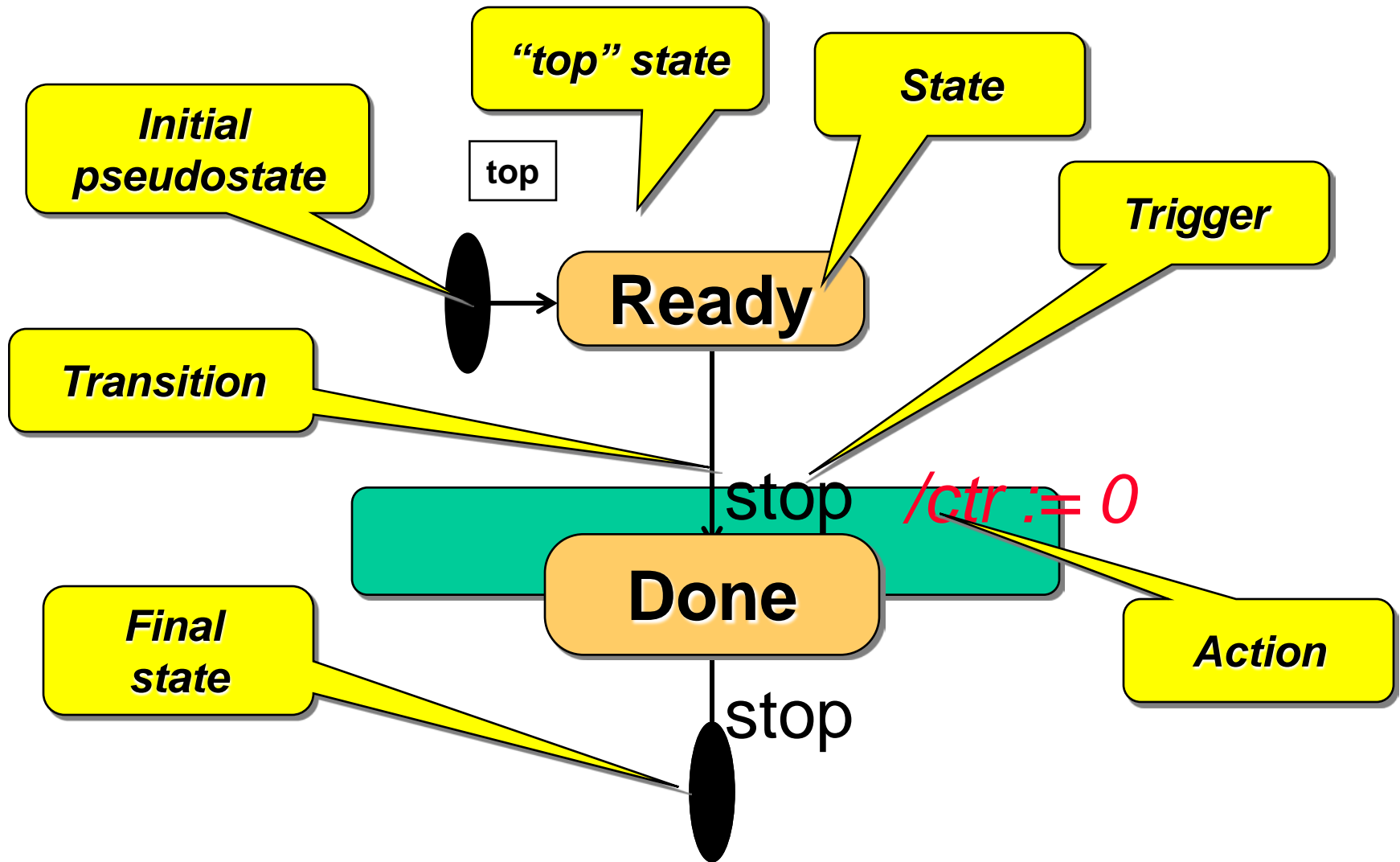
# Sequence Diagram



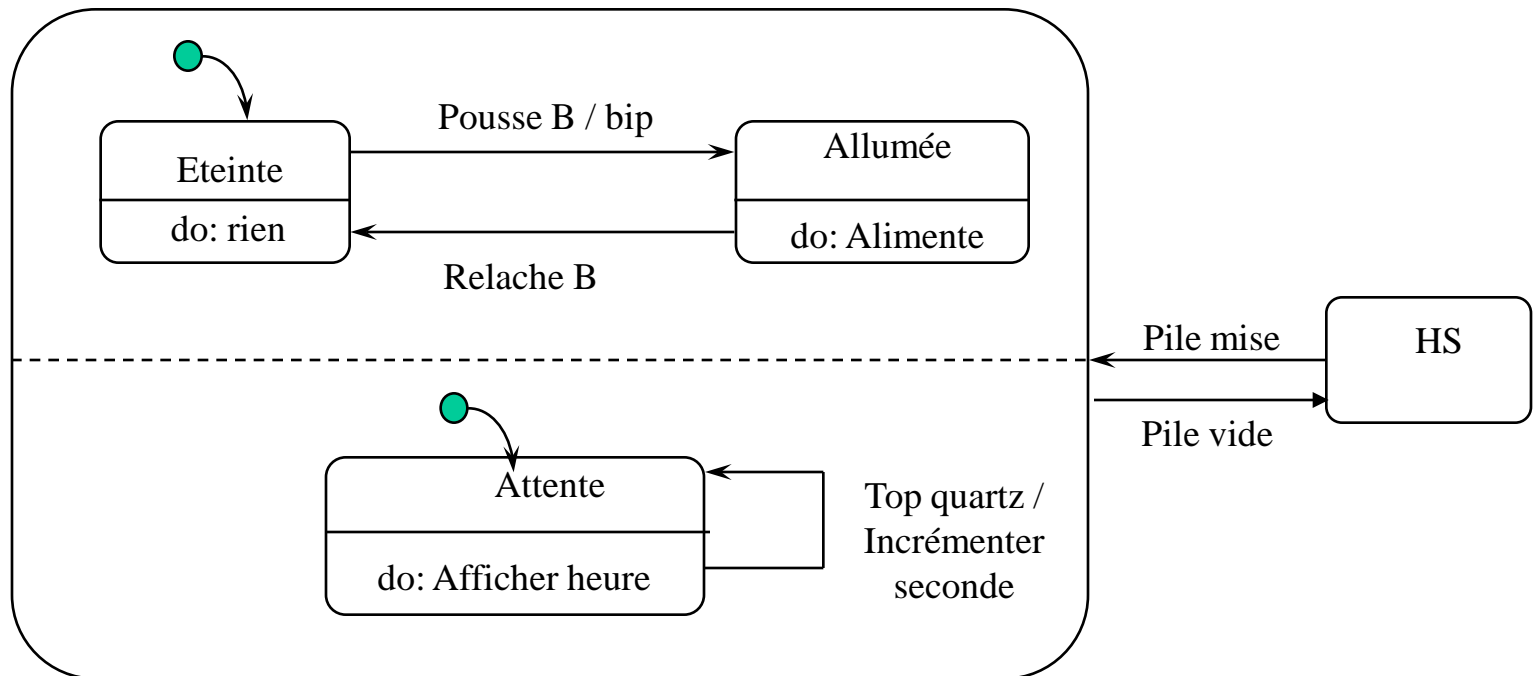
# Collaboration Diagram



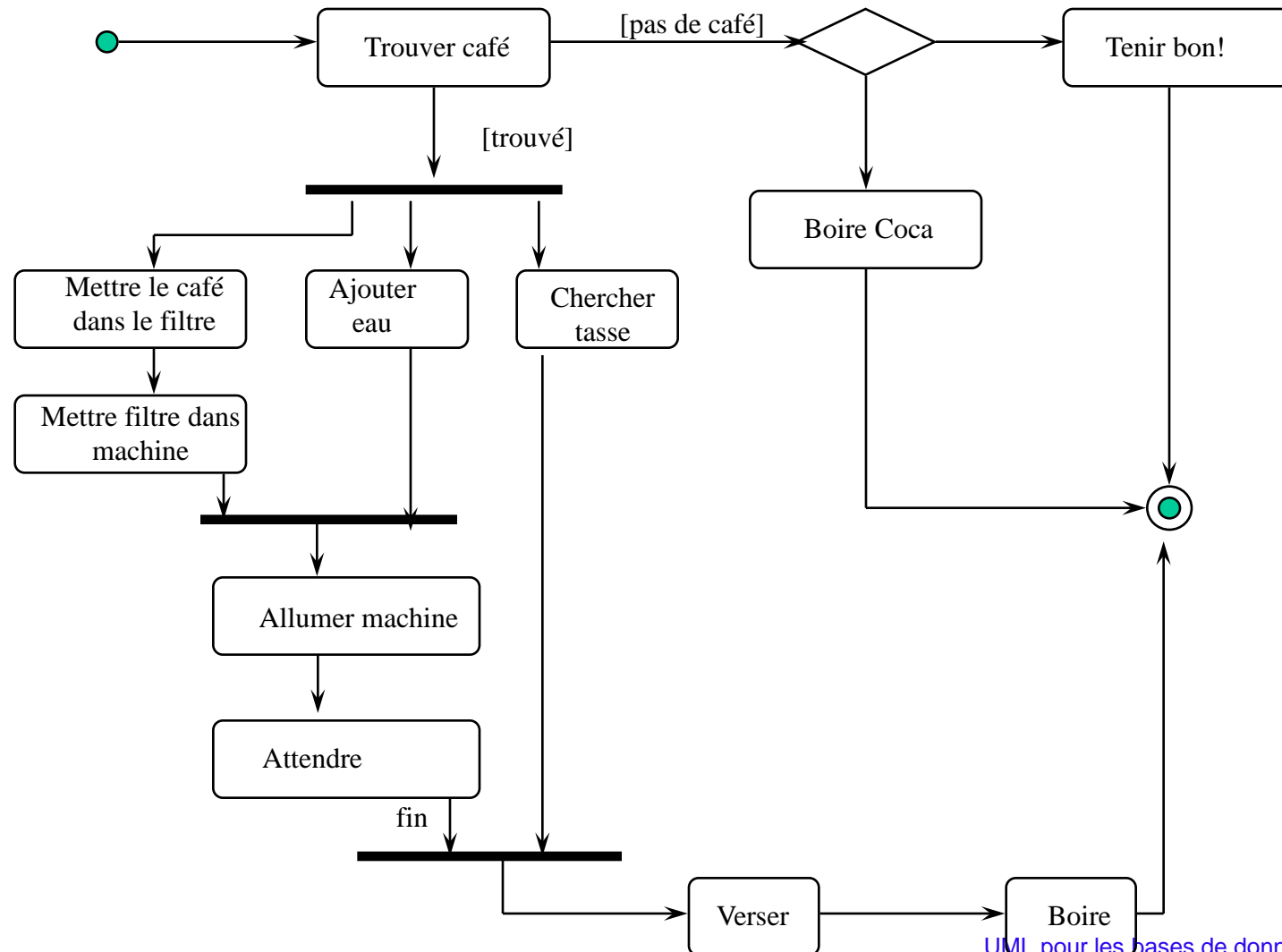
# Le diagramme Etats - Transitions



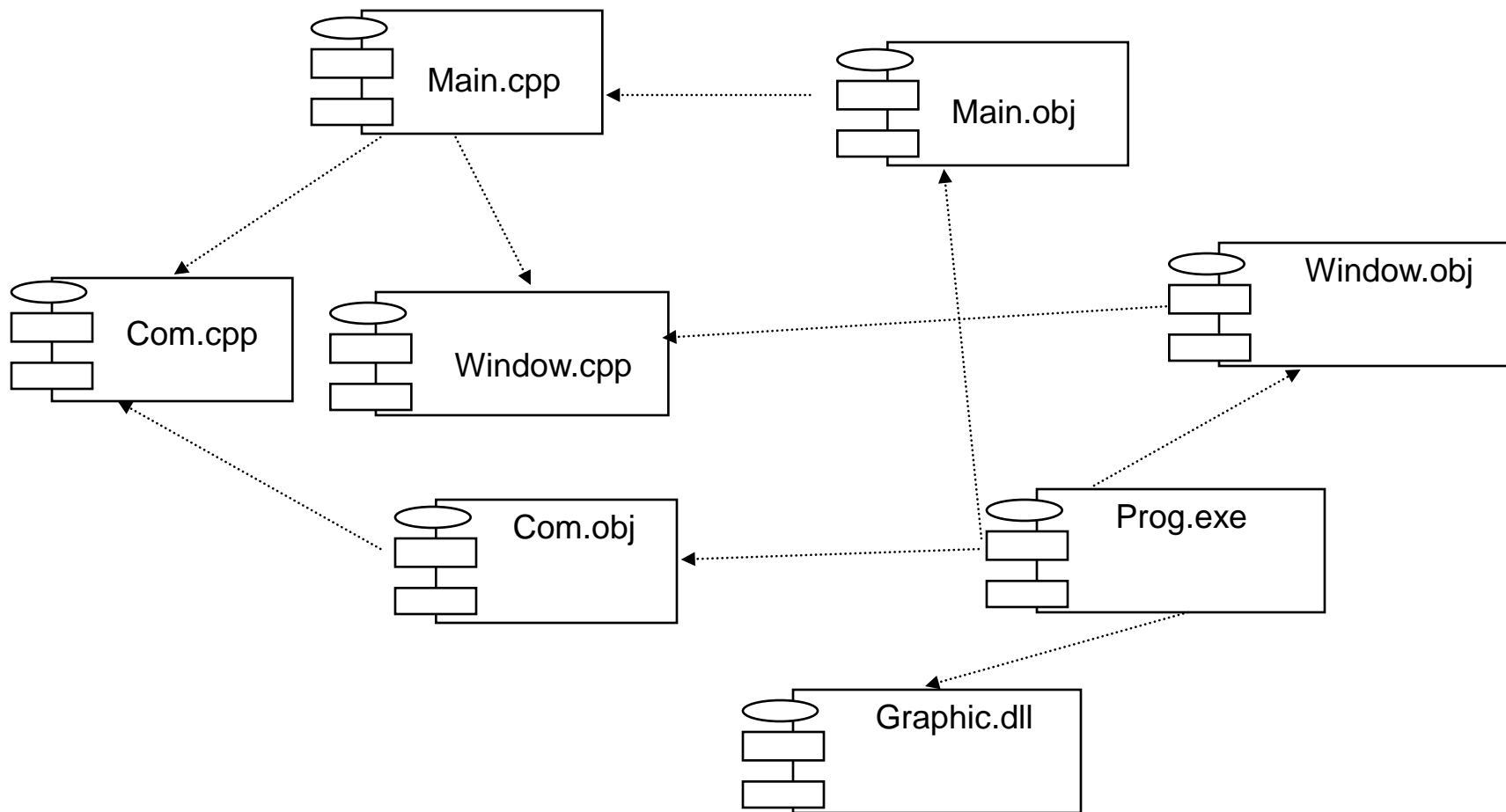
# Le diagramme Etats - Transitions



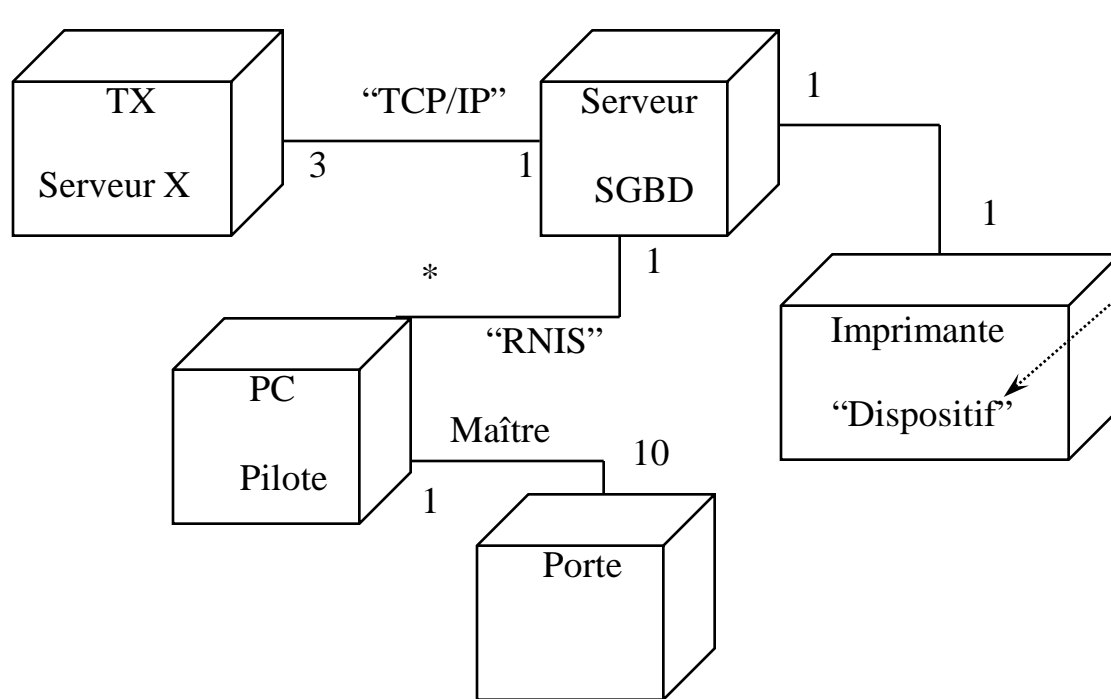
# Le diagramme d'activités



# Le diagrammes de composants



# Le diagrammes de déploiement



Processus hébergés

Chaque processus exécute un programme principal du même nom décrit dans le Diagramme de Composant

## 5. Une démarche

➤ *pilotée par les cas d 'utilisation*

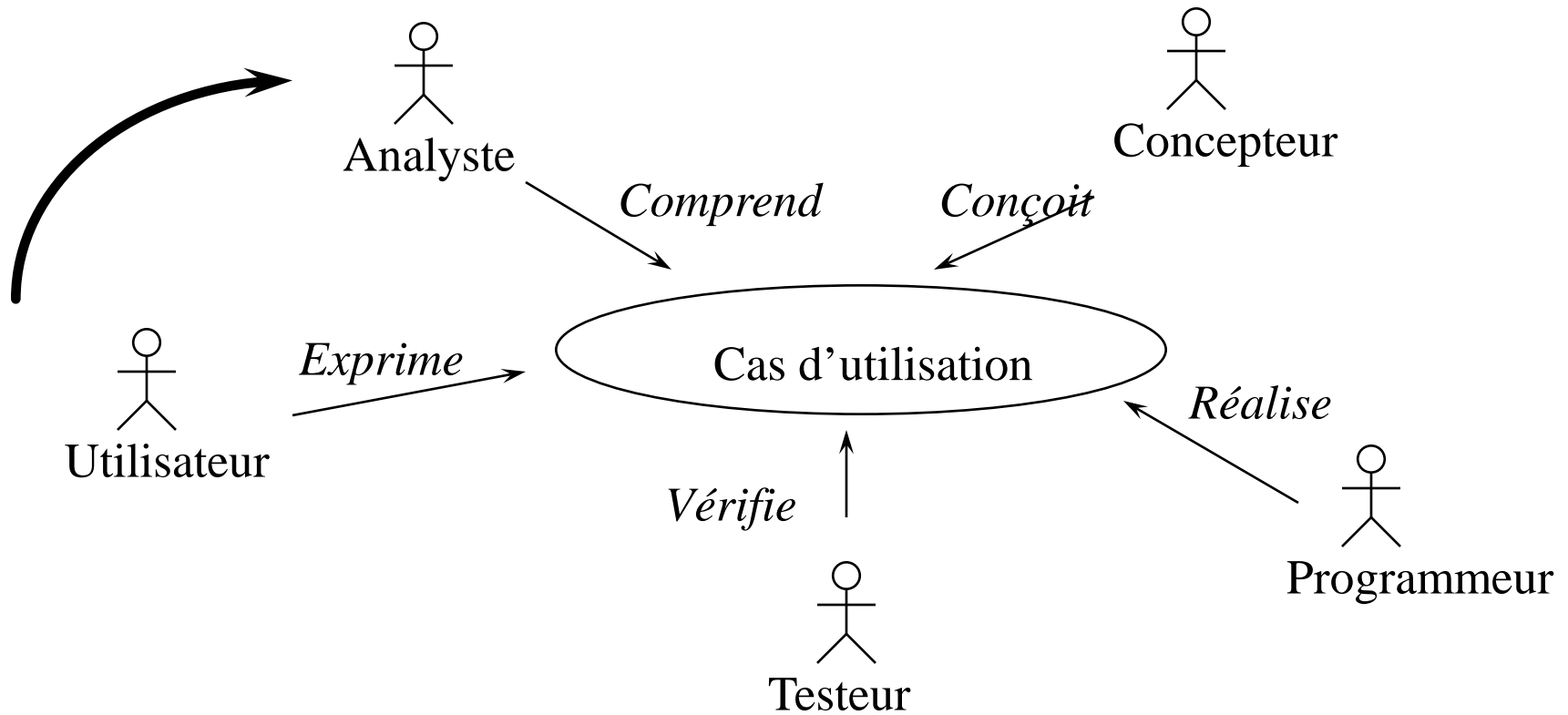
➤ *incrémentale*

➤ *itérative*

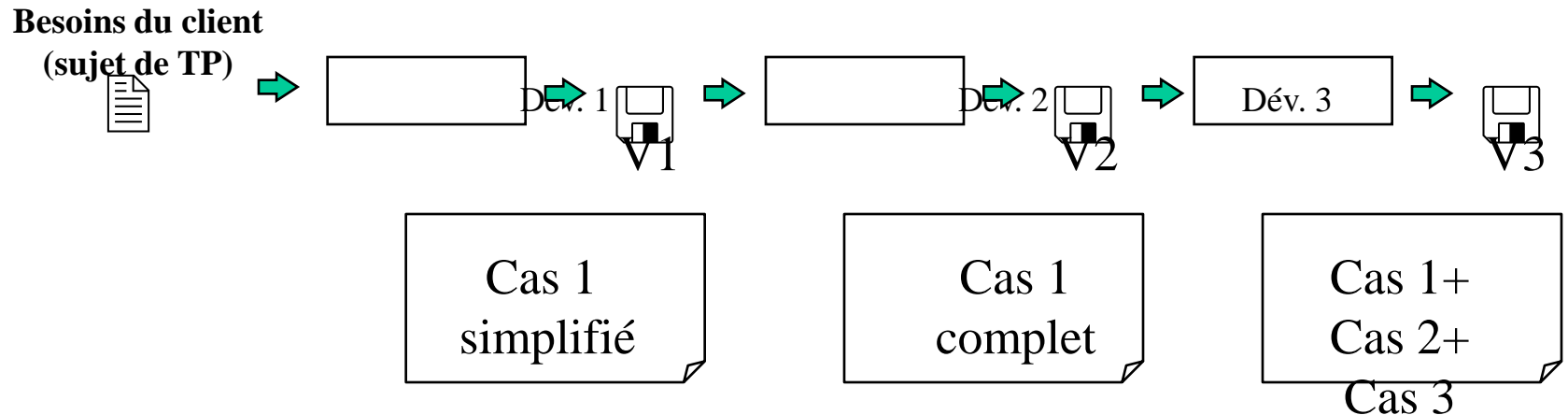
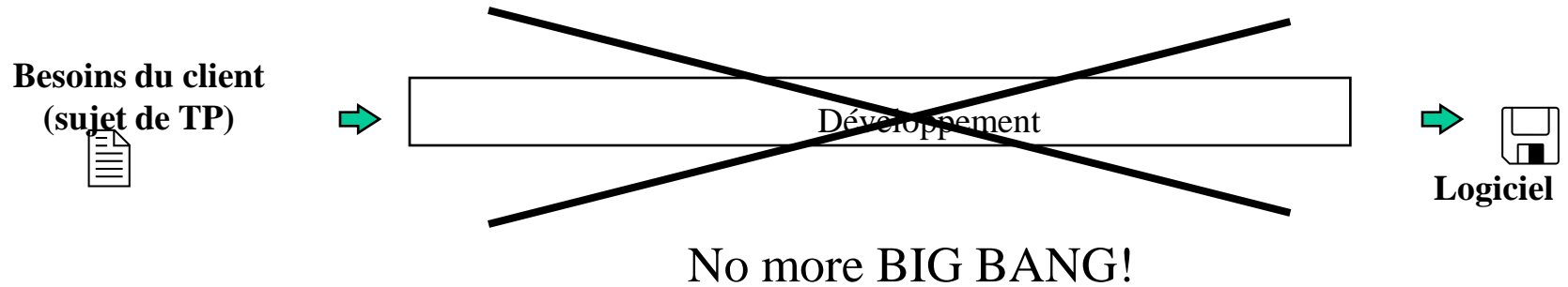
➤ *centrée sur l'architecture*



## Les cas d'utilisation pilotent le développement



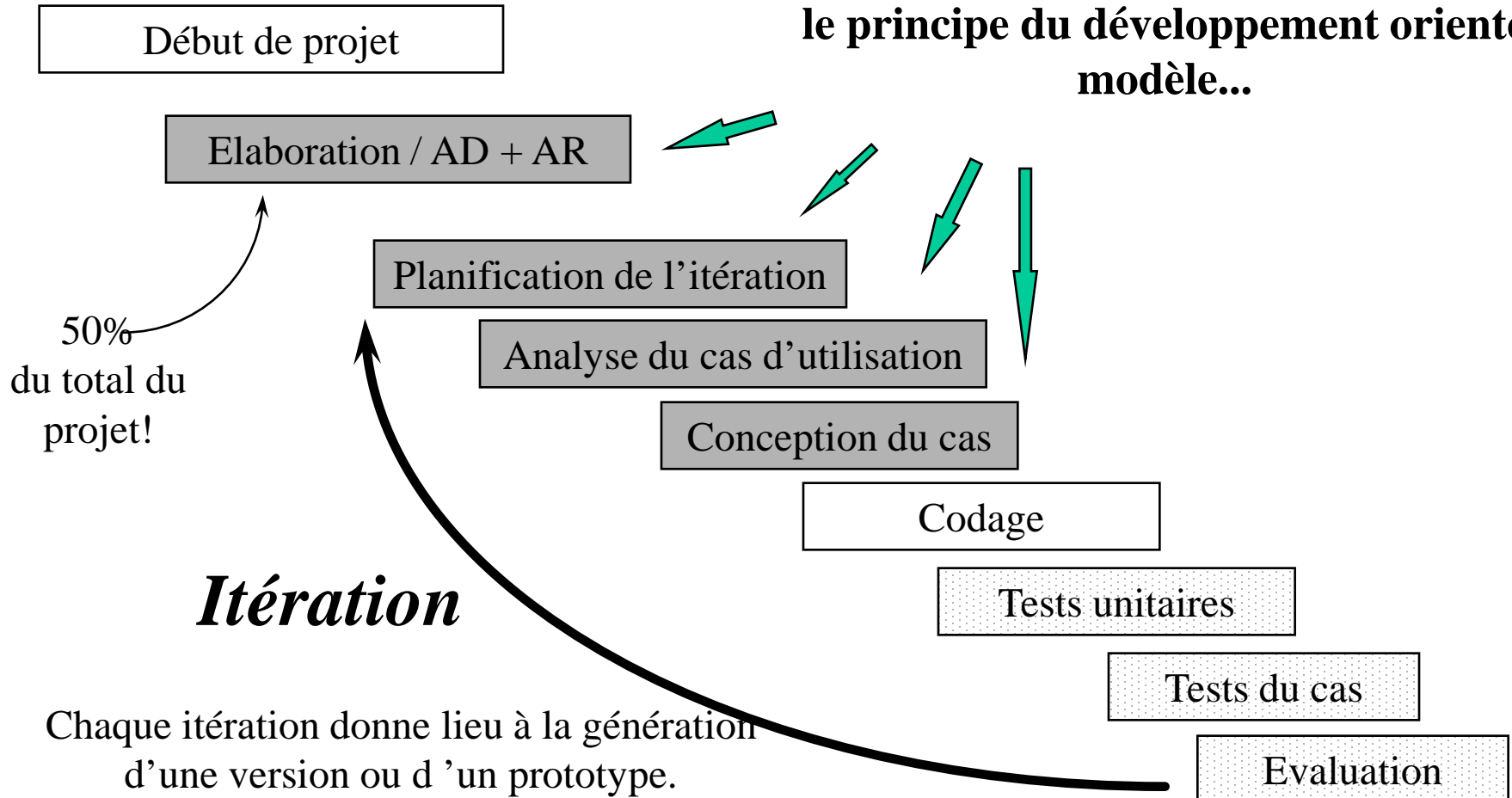
# *La construction est incrémentale*



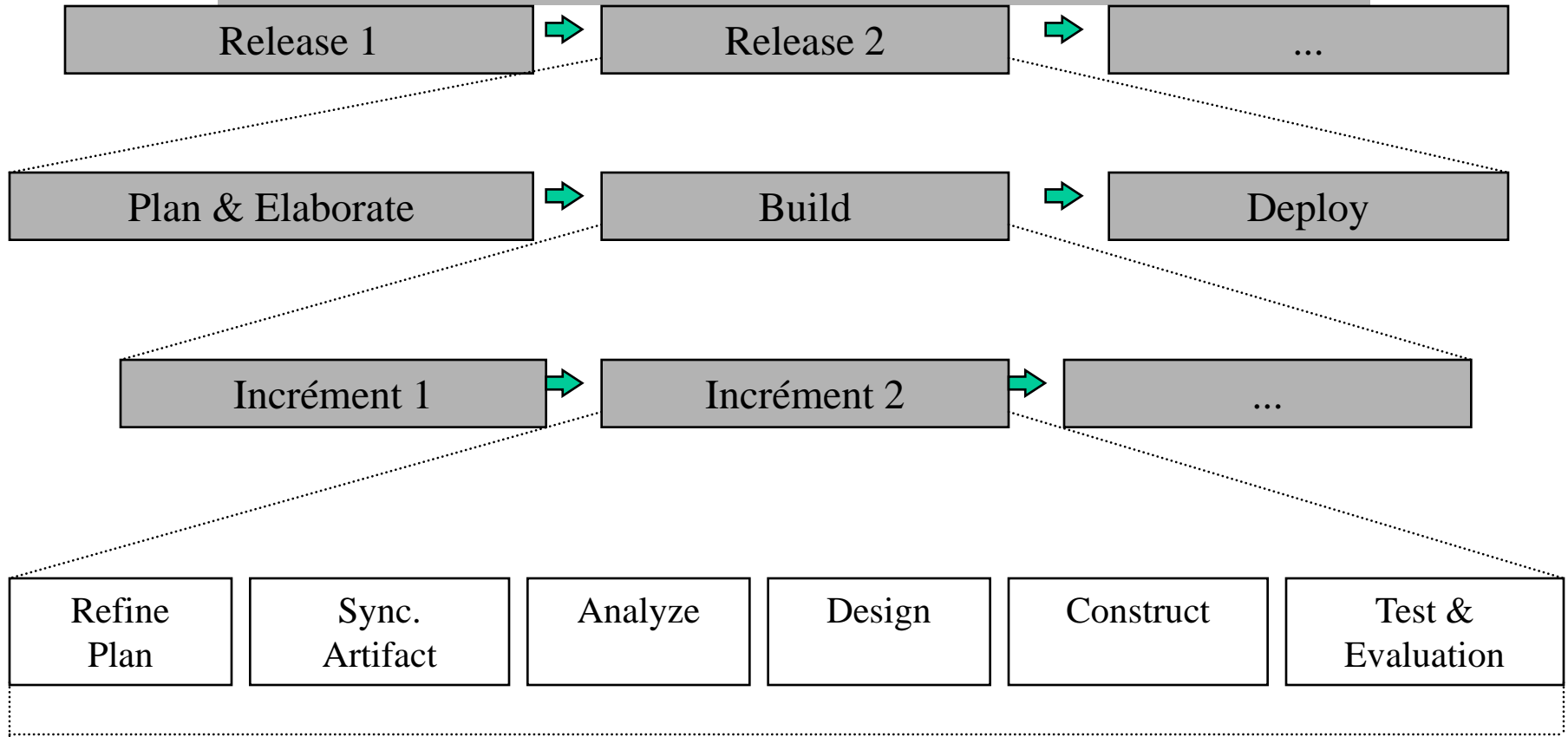
*Attention : tous les incréments ne donnent pas nécessairement lieu à un livrable client*

# Une construction itérative

Lors de chaque étape, on utilise un ou plusieurs des 9 diagrammes UML selon le principe du développement orienté modèle...



# *Le développement est itératif :* **Macro-processus / Micro processus**



*Time-Boxing (2 semaines à 2 mois)*

# ***Le développement est centré sur l'architecture***

- *Les premiers cycles de développement doivent avoir pour objectif la recherche d'une architecture logicielle maximisant la fiabilité, la maintenabilité, la portabilité, la réutilisabilité*
- *Dans les premières itérations on porte ainsi l'effort sur les parties du logiciel présentant le **plus haut niveau de risque** du point de vue architectural*



# MODELISATION, CONCEPTION SGBD

Michel Dubois

I.U.T. de Vannes

*Michel.Dubois@univ-ubs.fr*

# *HISTORIQUE DES MODÈLES DE DONNÉES*

mi-60 :

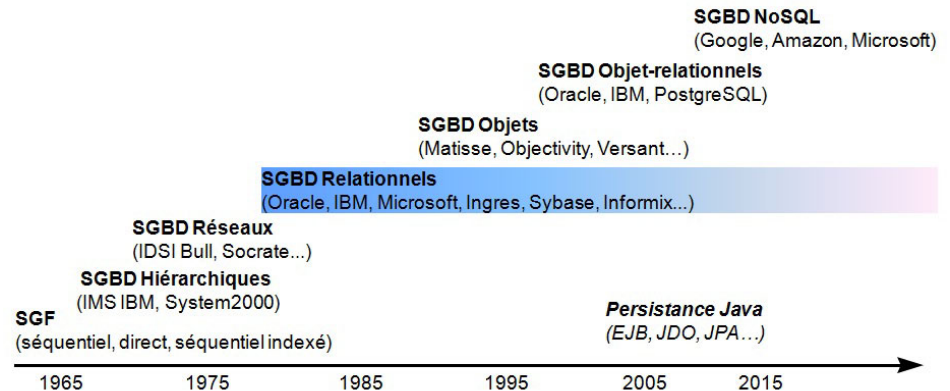
- HIERARCHIQUE IMS (IBM)
- RESEAU

70 :

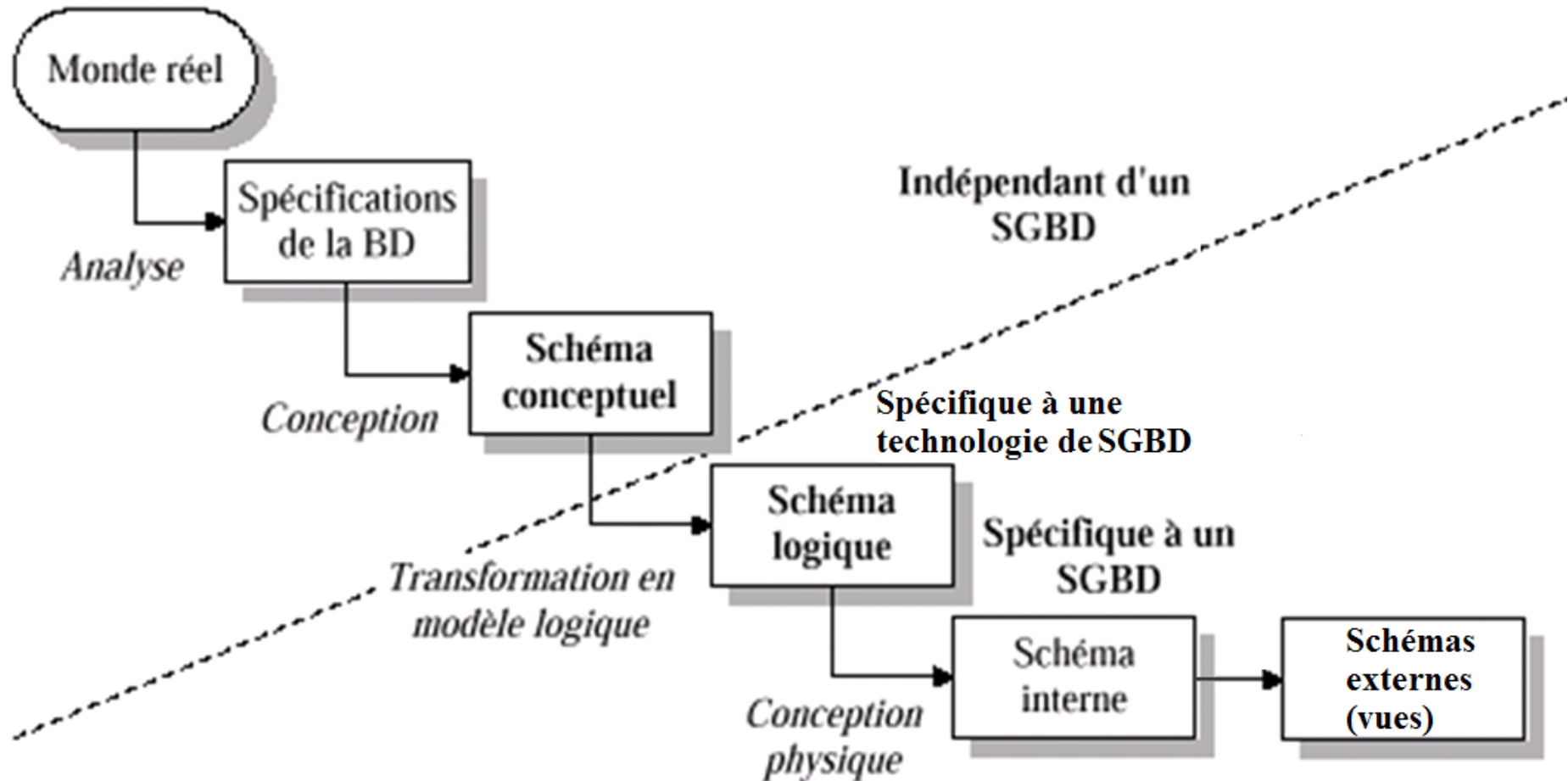
- RELATIONNEL
- ENTITE-RELATION

Fin 80 :

- ORIENTE-OBJETS
- RELATIONNEL ETENDU (Relationnel Objet)
- BD DEDUCTIVES



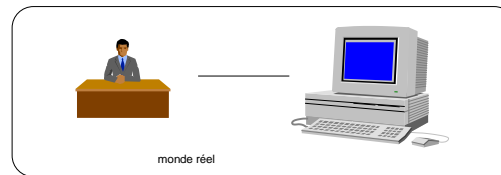
# PROCESSUS DE CONCEPTION





# DEMARCHE

# GENERALE

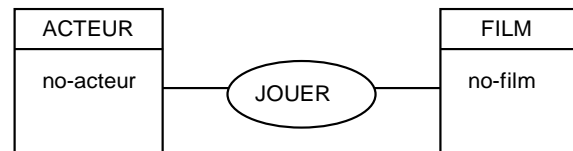


PERCEPTION



**MODELISATION CONCEPTUELLE**

DICTIONNAIRE DES DONNEES



**MODELISATION LOGIQUE**

ACTEUR (no-acteur, ...)  
FILM (no-film, ...)  
JOUER (no-acteur, no-film)

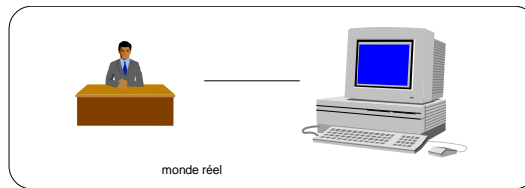


**UTILISATION D'UN SGBDR**

Create Table ACTEUR...  
Create Index...



*DEMARCHE*



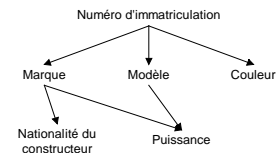
*DIRECTE*

PERCEPTION



**MODELISATION CONCEPTUELLE**

DICTIONNAIRE DES DONNEES  
GRAPHES DES DEPENDANCES FONCTIONNELLE  
OU  
NORMALISATION A PARTIR DE LA RELATION UNIVERSELLE



**MODELISATION LOGIQUE**

ACTEUR (no-acteur, ...)  
FILM (no-film, ...)  
JOUER (no-acteur, no-film)



**UTILISATION D'UN SGBDR**

Create Table ACTEUR...  
Create Index...



# *TERMINOLOGIE*

table  $\Leftrightarrow$  relation

enregistrement  $\Leftrightarrow$  occurrence  $\Leftrightarrow$  ligne de table  
 $\Leftrightarrow$  n-uplet  $\Leftrightarrow$  tuple

champ  $\Leftrightarrow$  attribut  $\Leftrightarrow$  colonne de table  $\Leftrightarrow$  donnée

intention d 'une relation  $\Leftrightarrow$  schéma d 'une relation

extension d 'une relation  $\Leftrightarrow$  contenu d 'une relation



# Chapitre 2

## Rappels

# Dictionnaire des données

Michel Dubois

I.U.T. de Vannes

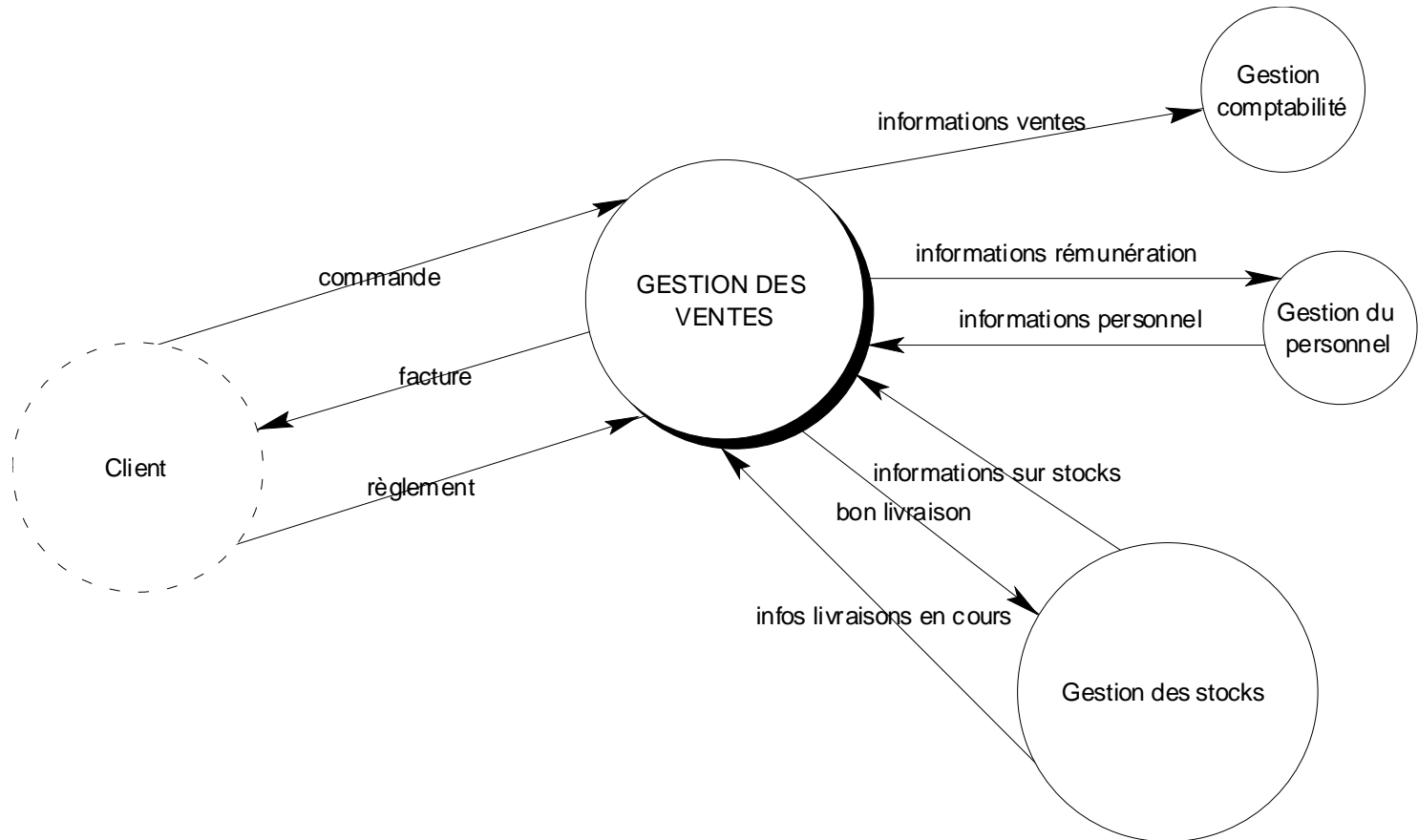
[\*Michel.Dubois@univ-ubs.fr\*](mailto:Michel.Dubois@univ-ubs.fr)

[Retour au plan](#)

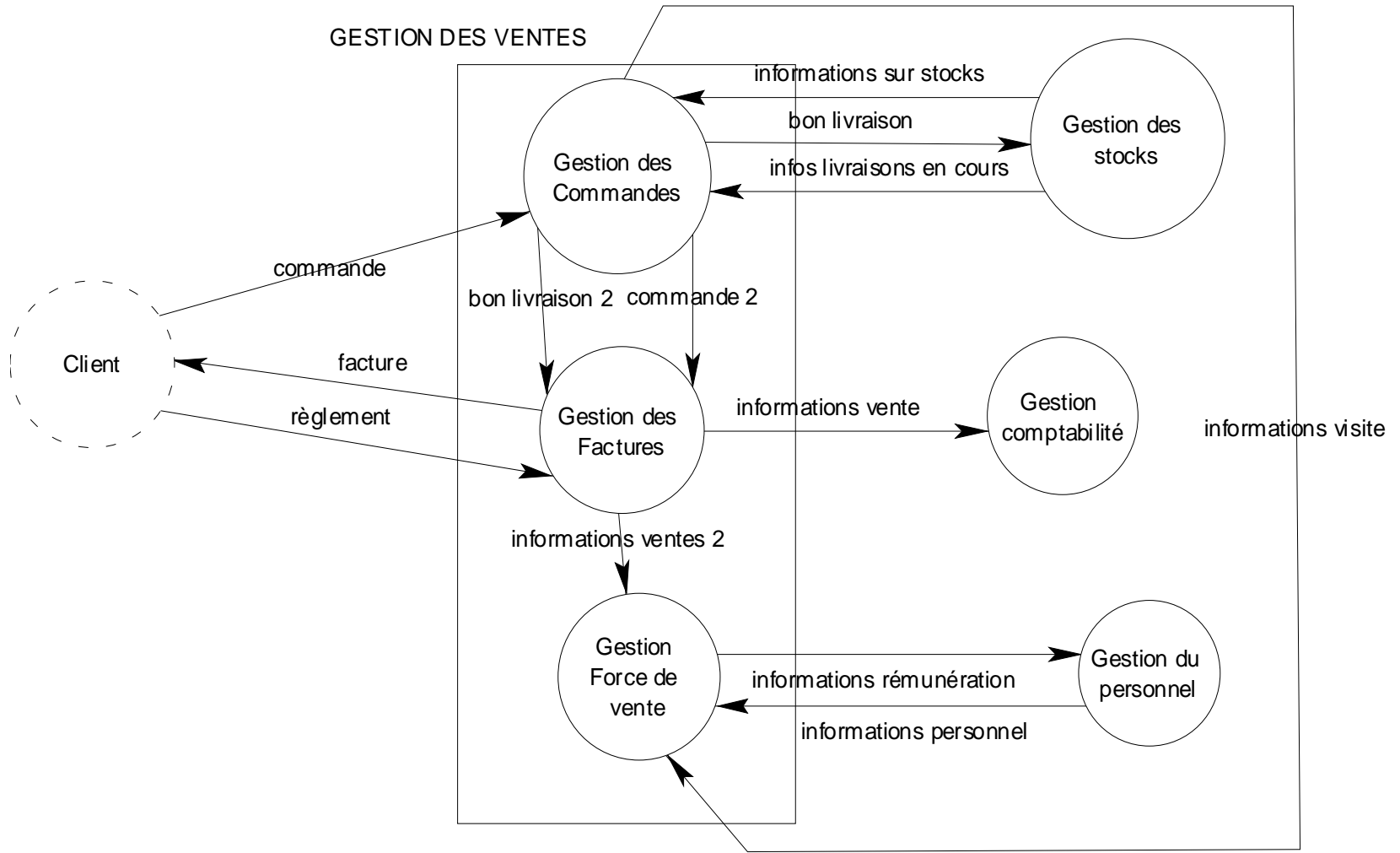
# *DOMAINE D'ÉTUDE*

- Le système à informatiser se confond rarement avec l'ensemble du Système d'Information. Il correspond généralement qu'à une de ses parties.
- Le domaine d'étude correspond à la partie du S.I. à informatiser.
- Les différentes parties du S.I. échangent des informations.
- Le S.I. échange aussi des informations avec des acteurs externes.
- Pour faciliter l'informatisation, on peut décomposer le domaine en sous domaine.

# *EXEMPLE DE DOMAINE*



# EXEMPLE DE SOUS DOMAINE



# *LE CONCEPT DE DONNÉE*

- Un système informatique traite essentiellement des données, c'est-à-dire des valeurs qui décrivent des phénomènes de la réalité et à partir desquels on peut obtenir des informations.
- Les données sont représentées à l'aide de nombres et de chaînes de caractères.
- Toute donnée est définie par :
  - un identificateur ou nom,
  - un type,
  - une valeur.



# Exemple de *DONNÉES*

Monsieur LEBAS est né le 24.01.58 ; il habite à Toulouse dans l'arrondissement 31400.

identificateur	nom
type	chaîne de caractères
valeur	'LEBAS'

identificateur	date de naissance
type	date
valeur	24.01.58

identificateur	code postal
type	entier
valeur	31400

identificateur	ville
type	chaîne de caractères
valeur	'TOULOUSE'

# *TYPE D'UNE DONNÉE*

- Il spécifie :
  - l'ensemble des valeurs que peut prendre la donnée,
  - l'ensemble des opérations applicables à ces valeurs.
- On retiendra les types neutres suivants:
  - Type chaîne ou texte
  - Type numérique / entier
  - Type numérique / reel
  - Type monétaire
  - Type logique Oui/Non
  - Type date

# *LE DILEMME ENTRE LES TYPES NUMÉRIQUES ET TEXTES*

types numériques :

- + généralement plus économes en espace de stockage
- incapables de recevoir des valeurs non numériques
- limite pour les chiffres significatifs

le type Texte :

- + peut très bien recevoir des valeurs numériques, sous la forme de suite de chiffres
- impossibilité des calculs arithmétiques (somme, moyenne)
- ne contrôle pas si la valeur est numérique

# *NE PAS CONFONDRE DONNÉES ET VALEURS*

- Une erreur souvent commise est de prendre la valeur d'une donnée pour une donnée propre.

Le formulaire suivant, contenant 5 cases à cocher, donnera lieu à la spécification de 3 données seulement.

Une donnée Pot Catalytique de type Oui/Non, une donnée ABS de type Oui/Non et une donnée Airbags de type texte limité aux valeurs ("Non"; "Conducteur"; "Conducteur et passager") ou fondée sur le type numérique limité aux valeurs ( 1 ; 2 ; 3 ).

En effet, les cases à cocher de la question des *Airbags* sont mutuellement exclusives et ne forment chacune que l'une des valeurs possibles de la donnée *Airbag*. Il serait maladroit de les assimiler chacune à une donnée indépendante puisqu'elles ne sont pas indépendantes.

*Quelles sont les caractéristiques de votre voiture :*

Munie d'un pot catalytique :

☒

Munie d'un système ABS de freinage :

☐

Munie d'Airbags :

☐ Non

☐ Côté conducteur

☒ Côté conducteur et passager

# *DONNÉE ÉLÉMENTAIRE OU COMPOSÉE*

- Une même donnée peut être perçue comme élémentaire (c'est-à-dire non décomposable) par un utilisateur et comme composée (de données élémentaires) par un autre.
- Exemples de données élémentaires :

identificateur	nom
type	chaîne de caractères
valeur	'LEBAS'

identificateur	adresse
type	chaîne de caractères
valeur	'30 rue Mozart 31400 TOULOUSE'

# *EXEMPLE DE DONNÉE COMPOSÉE*

adresse est une donnée composée de

identificateur	adresse rue
type	chaîne de caractères
valeur	'30 rue Mozart'

identificateur	code postal
type	entier
valeur	31400

identificateur	ville
type	chaîne de caractères
valeur	'TOULOUSE'

# *LE CHOIX DU NIVEAU D'AGRÉGATION DES DONNÉES*

- Vous devez mémoriser une adresse, quelle option retenez vous ? (adresse élémentaire ou adresse composée)
- Le choix de facilité consiste à retenir la première option. Toutefois, lorsque l'on fait ce choix, on rend délicates les recherches futures sur un composant de l'adresse.
- Ainsi, si on anticipe que l'on aura besoin pour des traitements ultérieurs d'accéder à un composant particulier de l'adresse, par exemple accéder à la ville pour réaliser un tri des personnes par ville de résidence, alors il faudra choisir un niveau de précision qui permet d'accéder individuellement à l'information demandée.

# *DICTIONNAIRE DES DONNÉES*

C'est la liste des données d'intérêt pour le domaine étudié.

A chaque fois que se présente une nouvelle information, le concepteur l'ajoute à la liste après avoir vérifié deux choses :

- Qu'elle ne soit pas déjà répertoriée dans la liste, sous le même libellé ou avec un synonyme. Il s'assure ainsi que la liste ne comporte pas de redondances.
- Qu'il n'existe pas une autre information dans la liste ayant la même appellation pour un sens différent (homonyme). Dans ce cas, il faudra pour éviter toute ambiguïté, donner des noms différents à chacune des informations ayant des sens différents.



# *DICTIONNAIRE DES DONNÉES*

- Le concepteur de la base de données doit dresser une liste d'information devant être manipulées par la base de données. Ces informations sont extraites de documents utilisés au sein du système préexistant ou de ses entretiens avec les utilisateurs futurs de la base de données.
- Le dictionnaire des données est un document destiné à recenser toutes les informations élémentaires devant constituer une base de données.
- On ne retiendra que les informations effectivement mémorisées ou utilisées par la base de données et ayant un intérêt pour les utilisateurs de la base de données.



# *DICTIONNAIRE DES DONNÉES ET UML*

La démarche pour alimenter le dictionnaire de données, après le diagramme synoptique des cas d'utilisation et l'étude en scénarios de chacun des cas, est :

1. Parcourir tous les scénarios des cas d'utilisation et faire l'inventaire des documents mentionnés;
2. Établir la structure de données de chaque document sous la forme d'un descriptif du document (voir transparents suivants)
3. À la lecture des scénarios, relever les données générées dans le cadre d'une activité qui ne sont pas nécessairement présentes dans les documents identifiés dans les scénarios;
4. Déterminer enfin, parmi les données présentes dans les documents ou les données qui sont générées par le système d'information mais non présentes dans les documents mentionnés, celles dont on devra assurer la persistance dans le contexte où une application de base de données serait réalisée pour automatiser les fonctions du système d'information.



# DESCRIPTIF DE DOCUMENT

Le descriptif du document permet de voir d'un coup d'œil les données présentes dans le document ainsi que les *structures de données*. Le modélisateur avisé doit pouvoir distinguer les données élémentaires et les groupes de données. Pour chaque groupe de données bien identifié, il devra créer une structure de données. La structure devra porter un nom et elle comportera une liste de données élémentaires. Une structure de données est inscrite en gras dans le descriptif. De plus, les données constituant la structure sont placées immédiatement au-dessous du nom de la structure avec un décalage vers la droite mettant bien en évidence le lien d'appartenance à la structure de données.

Une structure de données peut contenir une autre structure de données. On parle alors de structures *emboîtées*. Dans ce cas, le nom de la structure doit aussi être décalé vers la droite par rapport à la structure à laquelle il appartient, tout comme les autres données élémentaires. De plus, une structure peut être répétée dans un document. Un astérisque est alors inscrit entre parenthèses à la suite du nom de la structure.

Le descriptif d'un document est particulièrement utile au modélisateur. Non seulement permet-il une identification sans ambiguïté des données à modéliser, il reflète de plus les associations *un à plusieurs* ou *un à un* entre les données du domaine.

**Fiche client**

No client

2345678

Adresse

200, rue Albert  
Lévis, QC,  
Canada G6V 8R9

Nom client

A. Rioux Inc.

Limite crédit

20 000 \$

Représentant

Pierrette Proulx

Téléphone

418.833.8801

Télécopieur

418.833.8802

**Descriptif du document: Fiche client**

Numéro client

Nom client

Limite de crédit

**Adresse client**

Numéro civique

Rue

Ville

État ou province

Pays

Code postal

Représentant

Téléphone client

Télécopieur client

**ACME Québec Inc.**

**Facture**

No client

4523678

Adresse

1200, rue Omer Gouin  
Lévis, QC,  
Canada G6V 8R9

No facture

A2005

Nom client

Prolab Inc.

Date facture

23/12/05

Téléphone

418.833.8804

Télécopieur

418.833.8822

Commandes facturées	No produit	Nom	Quantité	Prix	Total
Commande: C677882	P10	Savon Plouf	350	1,50 \$	525,00 \$
	P48	Bain de minuit	180	2,00 \$	360,00 \$
	P35	Mousse douce	125	3,00 \$	375,00 \$
Commande: C877842	P48	Bain de minuit	200	2,00 \$	400,00 \$
	P37	Dentifrice doux	250	3,00 \$	705,00 \$
					Total : 2 365,00 \$

**Descriptif du document: Facture**

Date facture

Numéro facture

Numéro client

Nom client

**Adresse client**

Numéro civique

Rue

Ville

État ou Province

Pays

Code postal

Téléphone client

Télécopieur client

**Commande (\*)**

Numéro Commande

**Ligne commande (\*)**

Numéro produit

Nom produit

Quantité commandée

Prix unitaire

Prix total

Total facture



# DEMARCHE UML-BD

- Avant d'entreprendre la conception et la réalisation d'une application de base de données, le modélisateur peut compter sur 2 approches pour établir les besoins en matière de données :
  - Cas 1 : un document papier est à la source d'un projet de modélisation dans un domaine particulier. Nous en produirons d'abord un descriptif selon les conventions déjà exposées. Le modèle conceptuel / diagramme de classe ou la modélisation directe sera ensuite élaboré sur la base du descriptif et le cas échéant d'un certain nombre de règles de gestion qui viennent compléter le cas.
  - Cas 2 : débiter par la production d'un modèle de fonctionnement du système d'information ou débiter par la production d'un modèle de fonctionnement de l'application. Cette approche est appropriée dans la situation où l'application à réaliser ne peut s'appuyer sur un système d'information formel ou informel. Elle peut aussi être utilisée si on désire faire abstraction d'un système d'information existant. Cette décision peut se justifier en présence d'un système d'information bien réel mais devenu inapproprié, dépassé ou comportant de trop nombreuses lacunes. Il serait alors inutile de reproduire dans une application informatique les mêmes lacunes ou les mêmes carences, notamment au plan des données. On fait une étude de besoins en matière de données pour une application de base de données en débutant par le dessin d'esquisses de panoramas d'écrans. Le modélisateur s'entoure alors d'un nombre restreint d'utilisateurs directs ou indirects de l'application ou de personnes bien au fait des besoins de ces utilisateurs et sur lesquelles il peut compter. Ce n'est qu'après avoir réalisé plusieurs dessins de panoramas que modélisateur pourra s'attaquer à l'identification des cas d'utilisation et à rédaction des scénarios de cas d'utilisation, qui devront décrire comment l'utilisateur interagit avec l'application par le biais des panoramas d'écrans.
- Quelle que soit l'approche adoptée au début du processus de définition des besoins, le résultat sera le même: un recensement des données persistantes et leur représentation sous forme d'un dictionnaire de données.

# ESQUISSE DE PANORAMAS D'ECRANS

- Le modélisateur débute son travail en élaborant avec la direction du club et la personne en charge de l'administration du club des esquisses de panoramas d'écran. Il convient de dessiner en premier lieu le panorama utilisé pour assurer l'inscription des membres. Il existe de nombreux outils permettant de réaliser des documents électroniques ou des formulaires agissant comme interface utilisateur d'une application. Il suffit de mentionner Adobe Acrobat, Microsoft Visio ou le module Visual Basic de la suite Office de Microsoft. Ces outils permettent de dessiner des panoramas comportant une ou plusieurs fenêtres dotées de contrôles: boutons de commande, zones de texte pour la saisie, barres de défilement, étiquettes, formes géométriques, onglets, sélecteurs, etc.
- Le panorama représente le document électronique utilisé pour la saisie et la consultation des données sur les membres ainsi que pour l'exécution de certaines opérations: impression de la carte de membre ou ouverture d'une deuxième fenêtre faisant voir les données spécifiques à un membre athlète par exemple.
- Après avoir convenu ensemble de la pertinence de l'esquisse et de son contenu, le modélisateur dresse ensuite un descriptif du document électronique.

**Dossier membre**

Numéro: 1501

Prénom: Pauline

Nom: Dumoulin

NAS: 222-333-444 Date de naissance: 27-04-72

Adresse: 399 Allée des Plages  
Québec, Québec  
G3K-3A1

Téléphone(s): 418.636.1986 dom. 418.636.1986 p.1760 bur.

Année	Statut	Frais	Payé	Solde
1997	Athlète	25 \$	25 \$	0 \$
1998	Bénévole	0 \$		0 \$

Imprimer carte membre Transférer au système comptable

Consulter dossier bénévole Consulter dossier athlète

Rechercher membre

**Descriptif du document:** Dossier membre

Numéro membre  
Prénom membre  
Nom membre  
NAS membre  
Date de naissance membre  
Adresse membre  
Téléphone domicile  
Téléphone bureau  
Inscription(\*)  
Année inscription  
Statut membre (Athlète, bénévole)  
Frais inscription  
Montant payé  
Solde

# *RUBRIQUES DU DICTIONNAIRE DE DONNÉES*

Chaque donnée introduite dans le dictionnaire des données se voit précisée par une ligne dans un tableau ayant la forme suivante :

CODE	LIBELLE	NATURE	STABILITE	DOMAINE	COMMENTAIRE
...	...	...	...	...	...

CODE	LIBELLE	NATURE	STABILITE	DOMAINE	COMMENTAIRE
...	...	...	...	...	...

- Ces deux rubriques contiennent l'appellation conventionnelle de la donnée.
- Cette scission provient du fait que la plupart des SGBDR imposent des contraintes sur les noms de données.
- Par exemple ORACLE : les noms de données sont composés de 30 caractères maximum et ne peuvent être formés que d'un seul mot.
- La colonne CODE contiendra la représentation codée de la donnée, peu intelligible mais manipulable par le SGBDR, puis une colonne LIBELLE représentant l'appellation intelligible de la donnée.
- MS Access admet des noms de données sans trop de restrictions. Il suffit d'utiliser des crochets. La distinction n'est peut être pas nécessaire dans ce cas.

CODE	LIBELLE	NATURE	STABILITE	DOMAINE	COMMENTAIRE
...	...	...	...	...	...

Précise si la donnée est :

- **Non calculée** : La donnée ne peut être déduite par le système à partir d'autres données. Elle doit obligatoirement être saisie ou importée depuis une autre base de données.
- **Calculée** : La donnée est obtenue par l'application d'une opération sur une ou plusieurs autres données. Une telle donnée pourra être générée "au vol" par le SGBD et pourra donc ne pas être stockée dans la base de données.
- **Incrémentée** : Il s'agit d'une donnée, destinée à référencer un objet et dont la valeur est incrémentée (ajout de la valeur 1) à chaque fois qu'on crée une nouvelle occurrence dans la base de données.



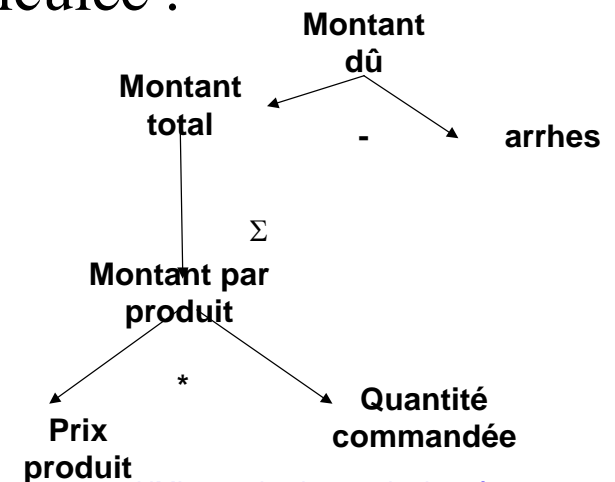
CODE	LIBELLE	NATURE	STABILITE	DOMAINE	COMMENTAIRE
...	...	...	...	...	...

## les données calculées :

Une telle donnée pourra être générée "au vol" par le SGBD et pourra donc ne pas être stockée dans la base de données. Toutefois, il faudra s'assurer que le dictionnaire des données comprenne toutes les données entrant dans son calcul. Dans le cas contraire, il faudra les ajouter.

Exemple de décomposition d'une donnée calculée :

On ne retiendra que les noeuds terminaux



CODE	LIBELLE	NATURE	STABILITE	DOMAINE	COMMENTAIRE
...	...	...	...	...	...

Cette colonne indique les possibilités de transformations subies par la donnée au cours du temps, une fois qu'elle a été saisie ou calculée. Une donnée peut être considérée comme :

- **Evolutive** : Elle est destinée à voir sa valeur modifiée au cours du temps.
- **Stable** : Une fois que l'on a mis une valeur dans cette donnée, elle ne changera pas au cours du temps.
- **Paramètre** : Il s'agit d'une donnée relativement stable, mais qui peut toutefois être modifiée de manière à faire évoluer le système, l'adapter à un changement de réglementation par exemple.

CODE	LIBELLE	NATURE	STABILITE	DOMAINE	COMMENTAIRE
...	...	...	...	...	...

## Les paramètres :

- Ils ne peuvent pas se rattacher à un objet-type ou un individu-type mais contiennent une valeur destinée à être exploitée par le système de base de données au cours de son fonctionnement (par exemple en tant que composante de calcul pour des données calculées). Une donnée de type paramètre n'existe qu'en un seul exemplaire pour tout le système.
- Si plusieurs occurrences d'un objet-type quelconque ou d'un individu-type quelconque peuvent avoir des valeurs différentes pour cette donnée, alors ce n'est pas un paramètre mais une donnée évolutive.

CODE	LIBELLE	NATURE	STABILITE	DOMAINE	COMMENTAIRE
...	...	...	...	...	...

- Le domaine décrit l'ensemble des valeurs pouvant être stockées dans une donnée.
- Le domaine de valeur sera renseigné tout d'abord par **un type de données** (ORACLE ou ACCESS ou neutre)
- Une **valeur par défaut**, qui correspondra à la valeur affectée à la donnée avant toute forme de saisie.
- Des **restrictions** ou **conditions de validité** : Une donnée *couleur des yeux* pourra être de type Texte[20 caractères] avec une restriction supplémentaire prenant la forme d'une liste de valeurs autorisées, un salaire **>0**.
- Un caractère de **saisie obligatoire** ou non. Ceci rend facultative/obligatoire la saisie d'une valeur.

CODE	LIBELLE	NATURE	STABILITE	DOMAINE	COMMENTAIRE
...	...	...	...	...	...

Cette colonne contient :

- un descriptif de chaque donnée (si nécessaire) ou même un exemple.
- la formule de calcul des données ayant la nature calculée (obligatoire). L'intérêt de faire apparaître la formule de calcul en clair permet de vérifier que les informations nécessaires à ce calcul sont bien présentes dans le dictionnaire des données (dans le cas contraire, il faut les rajouter).

# EXEMPLE DE DD

CODE	LIBELLE	NATURE	CATEGORIE	DOMAINE	COMMENTAIRE
NOM_FOUR	Nom du fournisseur	Non calculée / saisie	Stable	texte[40]	11
NUM_FOUR	Numéro d'un fournisseur	Incrémentée	Stable	numérique	Rajoutée du fait de la possibilité d'homonymie précédente afin de distinguer deux fournisseurs ayant le même nom.
RAIS_FOUR	Raison sociale du fournisseur	Non calculée / saisie	Stable	texte[30]	forme juridique+ inscription au registre du commerce
TEL_FOUR	Téléphone du fournisseur	Non calculée / saisie	Evolutive	numérique	
ADR_FOUR	Adresse du fournisseur	Non calculée / saisie	Evolutive	texte[100]	
REFARTICLE	Référence interne d'un article	Incrémentée	Stable	numérique	Unique, peut servir de clé
REF_FOUR	Référence d'un article chez le fournisseur	Non calculée / saisie	Stable	texte[10]	Présumée stable si fournisseur cohérent
PRIX_FOUR	Prix d'un article chez le fournisseur	Non calculée / saisie	Evolutive	numérique	
NOM_CLIENT	Nom d'un client	Non calculée / saisie	Stable	texte[20]	Hypothèse restrictive : Possibilité d'homonymie.
ADR_CLIENT	Adresse d'un client	Non calculée / saisie	Evolutive	texte[100]	
REF_CLIENT	Référence d'un client	Incrémentée	Stable	numérique	Unique, peut servir de clé
N_FACTURE	Numéro d'une facture	Incrémentée	Stable	numérique	Unique (législation), peut servir de clé
DATE_FACT	Date d'une facture	Non calculée / saisie	Stable	date	Aurait pu également être considérée comme calculée
DESARTICLE	Désignation d'un article	Non calculée / saisie	Stable	texte[25]	
PRIX_ART	Prix de vente d'un article	Non calculée / saisie	Evolutive	numérique	Prix interne d'un article
QUANTITE	Quantité d'un article	Non calculée / saisie	Stable	numérique	Quantité apparaissant sur une ligne de facture
REMISE	Remise sur un article	Non calculée /saisie	Stable	numérique	Remise sur un article en %
MONTANT_HT	Montant HT d'un article	Calculée	Stable	numérique	Montant HT d'une ligne de facture = PRIX_ART*QUANTITE*(1-REMISE). Stable car pas de mise à jour en cas d'évolution de PRIX_ART.
TOTAL_HT	Total HT d'une facture	Calculée	Stable	numérique	somme de MONTANT_HT
TVA	Tva sur facture	Calculée	Stable	numérique	TOTAL_HT*0,186
TOTAL_TTC	Total TTC d'une facture	Calculée	Stable	numérique	TOTAL_HT+TVA
REGLEMENT	Mode de règlement	Non calculée /saisie	Stable	numérique	1 = liquide ; 2= CCP; 3=Virement Postal; 4=Chèque bancaire; 5=C.B.; 6=Virement bancaire
MARGE	Marge sur un article	Calculée	Stable	numérique	PRIX_ART-PRIX_FOUR, calculée seulement sur demande

# *DD DE LA BASE LOCATION DE VOITURE*

<b>CODE</b>	<b>LIBELLE</b>	<b>NATURE</b>	<b>CATEGORIE</b>	<b>DOMAINE</b>	<b>COMMENTAIRE</b>
NUM_IMMA	Numéro d'immatriculation	Non calculée / saisie	Stable	texte[20] obligatoire	Identifiant de voiture
MARQUE	Marque	Non calculée / saisie	Stable	Texte[30] obligatoire	
MODELE	Modèle	Non calculée / saisie	Stable	Texte[30] obligatoire	
COULEUR	Couleur du véhicule	Non calculée / saisie	Evolutive	Texte[10] dans ( « Bleu », « Rouge », « Vert », « Noire »)	
NAT_CONS	Nationalité du constructeur	Non calculée / saisie	Stable	texte[20] obligatoire	
PUISSANCE	Puissance du moteur	Non calculée / saisie	Stable	Numérique Obligatoire	

# *LA MISE EN ÉVIDENCE DES RÈGLES DE GESTION*

- Les spécifications littéraires de l'activité à informatiser sous forme d'une base de données peuvent être confuses.
- Il est intéressant d'extraire les points essentiels concernant les propriétés des objets ou des individus à mémoriser ou leurs relations.
- Une règle de gestion est associée au niveau conceptuel. Elle décrit le « quoi ». Elle ne décrit ni le « ou, quand, qui » (règle d'organisation), ni comment (règle techniques).



# EXEMPLES DE RÈGLES DE GESTION

- « *Une référence de client est numérique et séquentielle. Le premier client a le n°1, le second a le n°2, etc.* » explicite la nature incrémentée de la donnée Numéro client et son type numérique
- « *Un client peut acheter plus d'un seul produit ou plusieurs exemplaires d'un même produit lors d'un même achat.* » spécifie que les données relatives aux produits ne dépendent pas fonctionnellement de la donnée Référence du client.
- « *Le taux de TVA appliqué aux produits est uniforme fixé à 19,6%.* » indique que la donnée Taux de TVA des produits sera un paramètre (du fait de la présence du mot uniforme)

# *LISTING DES RÈGLES DE GESTION*

- En faire un listing préalable comporte de nombreux avantages.

Il permet entre autres de :

- clarifier votre compréhension du problème par une réécriture dans un style précis de l'énoncé,
- mettre noir sur blanc les hypothèses que vous dressez sur des éventuels non-dits afin que vous vous y teniez par la suite,
- accélérer votre recherche d'information pour préciser un type de données ou une dépendance fonctionnelle.

# *LE TRAITEMENT DES DONNÉES CALCULÉES OU PARAMÈTRES*

- Les données calculées ou paramètres ne doivent pas figurer dans les tables, ni dans les matrices ou graphes des dépendances fonctionnelles. Elles sont cantonnées au dictionnaire des données. On se référera à ce document lorsque l'on désirera les utiliser.
- En effet, les données calculées peuvent être déduites d'autres données, aussi, elles ne sont pas stockées dans la base de données. Toutefois, lorsque le besoin d'obtenir leur valeur se fera sentir durant l'exploitation ultérieure de la base de données, on demandera au SGBDR de les calculer notamment à l'aide de vues. Cependant si le calcul est systématique, ou souvent demandé, une optimisation peut être de stocker les résultats.
- Les données paramètres seront directement inscrites dans les formules de calcul des données calculées, donc ne sont pas stockées dans les tables de la base de données. Une autre solution est de créer une table paramètre ayant qu'un seul tuple.



# Chapitre 3

## Schéma

# Entité Association

Michel Dubois

I.U.T. de Vannes

[Michel.Dubois@univ-ubs.fr](mailto:Michel.Dubois@univ-ubs.fr)

[Retour au plan](#)

# Entité

- Une entité est un objet du réel perçu qui a une existence propre. Elle peut donc être décrite indépendamment des autres objets.
- Par extension, une entité désigne l'ensemble des objets du réel perçu qui jouent le même rôle dans le système d'information et qui sont donc décrits par les mêmes caractéristiques.

# Entité

- Une entité peut être
  - un individu, comme par exemple : une personne, un client, un animal, ...
  - un objet concret : une matière première, un produit, ...
  - un objet abstrait : un service, un compte bancaire, ...
  - un lieu : une région, un magasin, ...
  - un objet documentaire : un contrat, une facture, ...

# Association

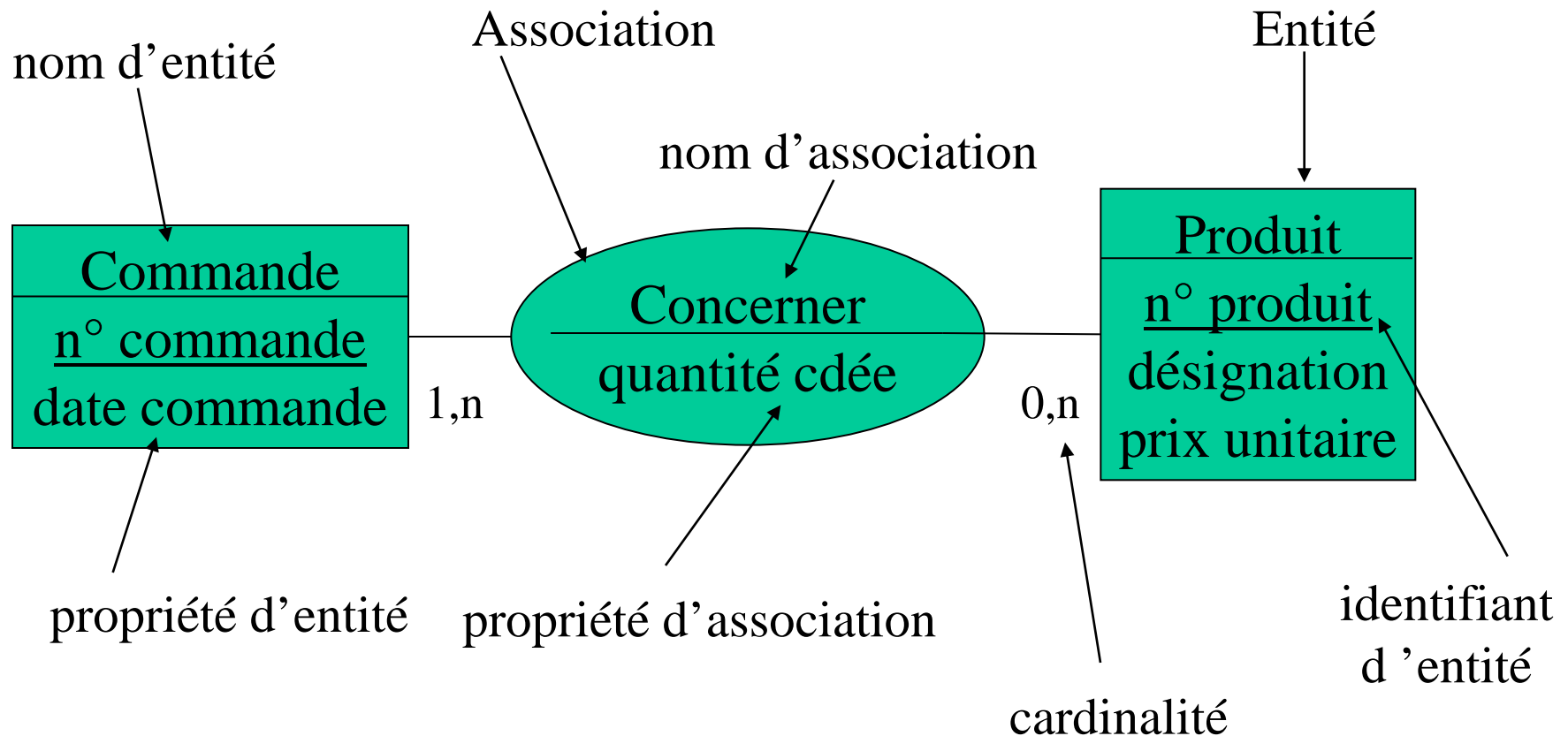
- lien sémantique entre au moins deux entités
- le lien n 'est pas orienté : « les commandes comportent des produits » veut également dire que « les produits peuvent être commandés ».
- souvent nommée par un verbe ou un substantif
- une association est un objet du réel perçu qui n 'a d 'existence qu 'au travers des entités qu 'elle met en correspondance
- par extension, une association désigne l 'ensemble des associations de même nature dans le système d 'information et qui sont donc décrites de la même façon.

# Propriété

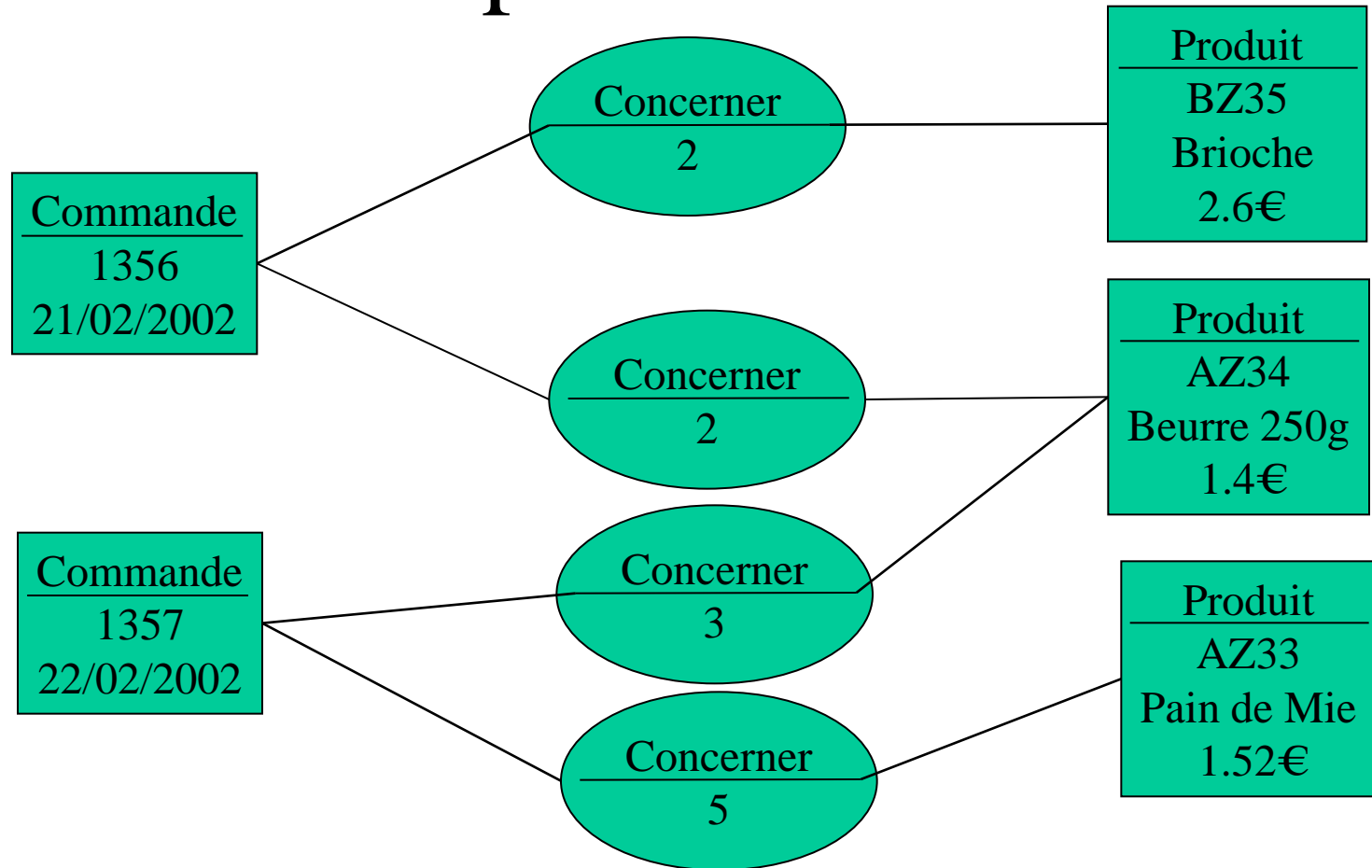
- donnée élémentaire permettant de décrire une entité ou une association
- cette donnée peut se mesurer par une valeur
- synonyme : attribut
- Règles de base :
  - une même propriété ne peut pas servir à décrire deux objets (entité / association) différents
  - une entité possède au moins une propriété, son identifiant
  - une association peut ne pas porter de propriété



# Représentation graphique



# Description exhaustive



# Occurrence

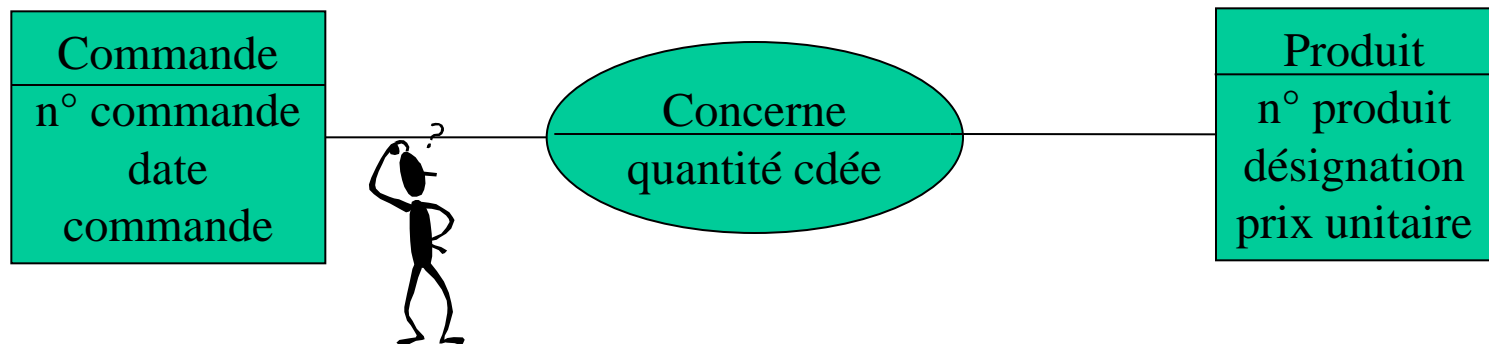
- Réalisation particulière d 'une entité, propriété ou association
- synonyme : instance
- Exemple d 'occurrence de l 'entité commande :  
n°1356 du 21/02/2002
- Exemple d 'occurrence de l 'association concerner :  
2 produits BZ35 pour la commande n°1356
- Une occurrence de propriété est une valeur :  
l 'occurrence de la propriété prix unitaire pour le  
produit BZ35 est 2.6 €

# Cardinalité

- La cardinalité est une notion obligatoire du modèle qui permet de résoudre la question de l'anomalie de la commande 1246 qui est une commande vide
- C'est donc l'expression d'une contrainte perçue sur le monde et que l'on indique dans le modèle. Par exemple : une commande concerne toujours au moins un produit.

# Calcul des cardinalités

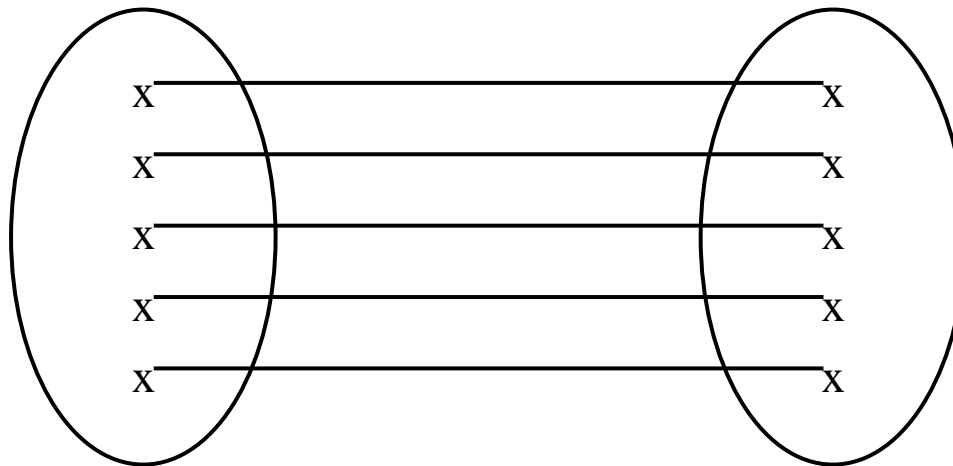
- Pour une occurrence de l'entité que j'examine, combien y a-t-il d'occurrences possibles de l'association à laquelle elle participe, au plus et au moins ?
- Pour calculer la cardinalité, se positionner (virtuellement) sur l'entité concernée et regarder en face le nombre de liens qu'une occurrence peut avoir avec l'association.
- Faire la même chose pour l'ensemble des entités qui participent à la relation



# *LES ASSOCIATIONS DE TYPE 1-1*

Exemple :

FOURNISSEUR      (fournir)      PRODUIT

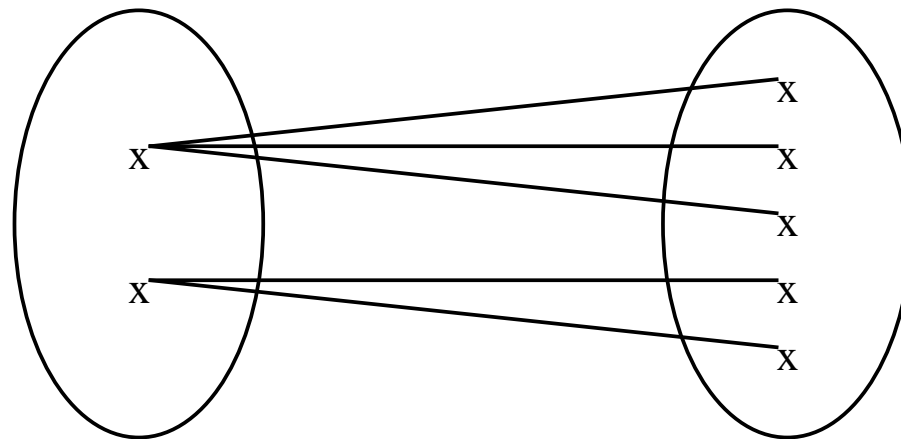


Tout fournisseur ne fournit qu'un seul produit.  
Tout produit n'est fourni que par un seul fournisseur.

# *LES ASSOCIATIONS DE TYPE 1-N*

Exemple :

FOURNISSEURS (fournir) PRODUITS

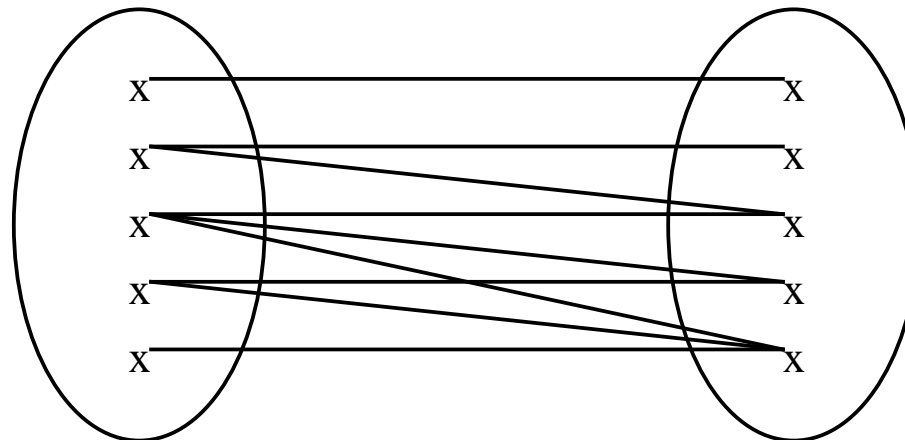


Un fournisseur fournit un ou plusieurs produits.  
Tout produit n'est fourni que par un seul fournisseur.

# *LES ASSOCIATIONS DE TYPE M-N*

Exemple :

FOURNISSEURS      (fournir)      PRODUITS



Un fournisseur fournit un ou plusieurs produits.  
Un produit est fourni par un ou plusieurs fournisseurs.



# *CARDINALITES MINIMALES*

Valeur	Définition	Exemple
<b>0</b>	<b>Une occurrence de l'entité <u>peut exister</u> sans participer à l'association</b>	un produit peut ne pas être commandé
<b>1</b>	<b>Une occurrence de l'entité participe nécessairement au moins une fois à une occurrence d'association</b>	toute commande concerne au moins un produit

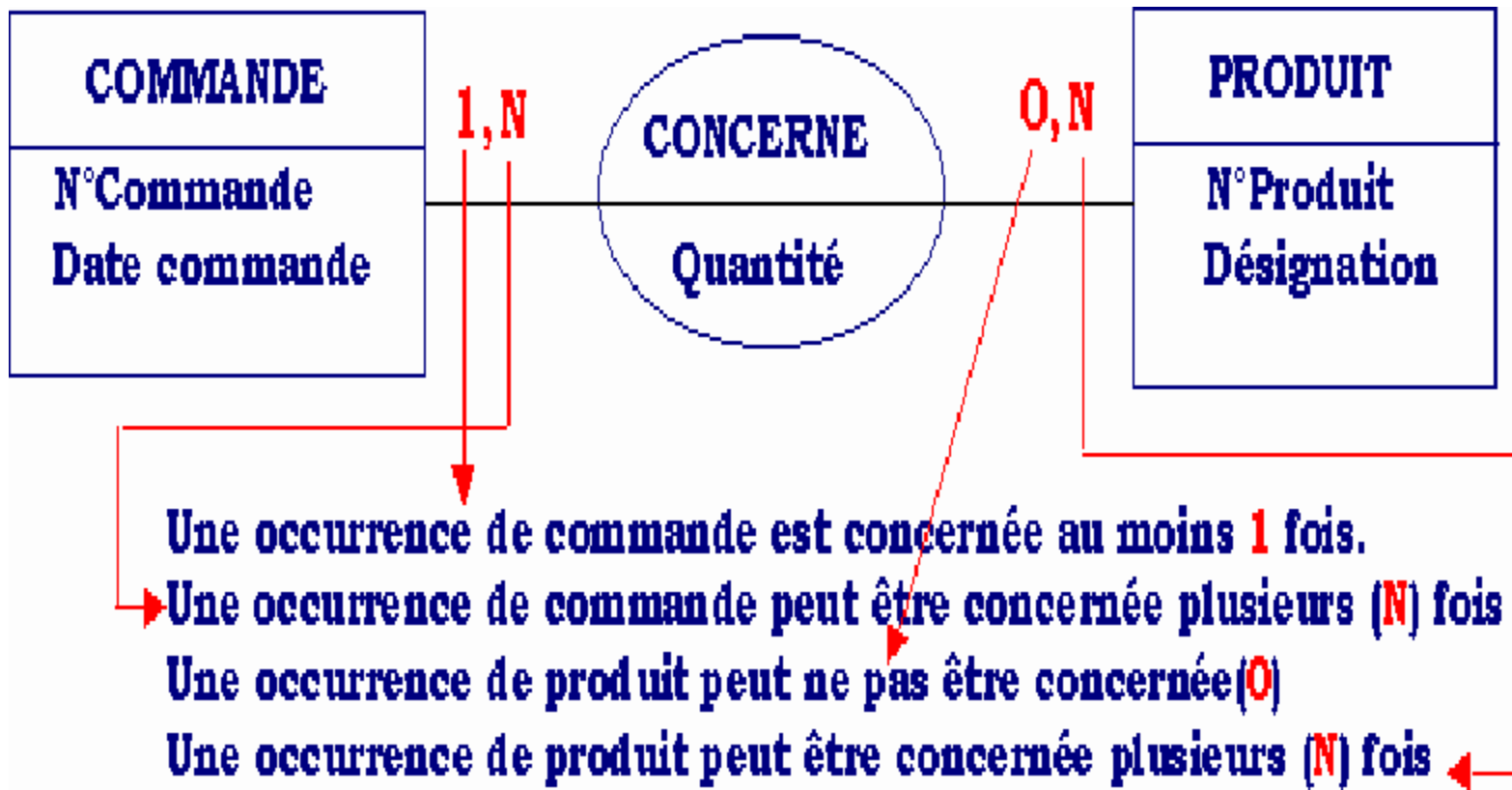
# *CARDINALITES MAXIMALES*

<b>Valeur</b>	<b>Définition</b>	<b>Exemple</b>
<b>1</b>	<b>Une occurrence de l'entité participe au plus une fois</b>	un employé travaille au plus dans un service
<b>N</b>	<b>Une occurrence de l'entité peut participer plusieurs fois</b>	une commande peut concerner plusieurs produits

# Configurations possibles

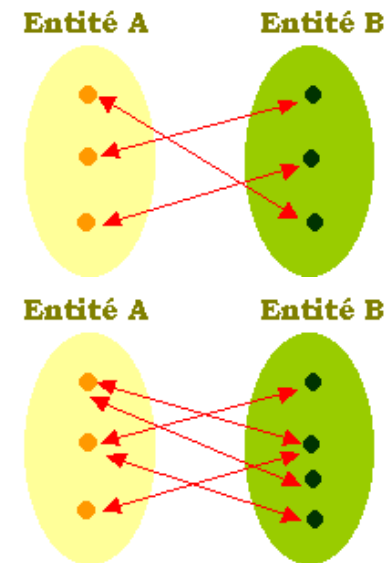
- 0,1 : une occurrence de l'entité participe au plus une fois à l'association
- 1,1 : une occurrence de l'entité participe exactement une fois à l'association
- 0,N : une occurrence de l'entité peut ne pas participer à l'association
- 1,N : une occurrence de l'entité participe au moins une fois à l'association

# EXEMPLE



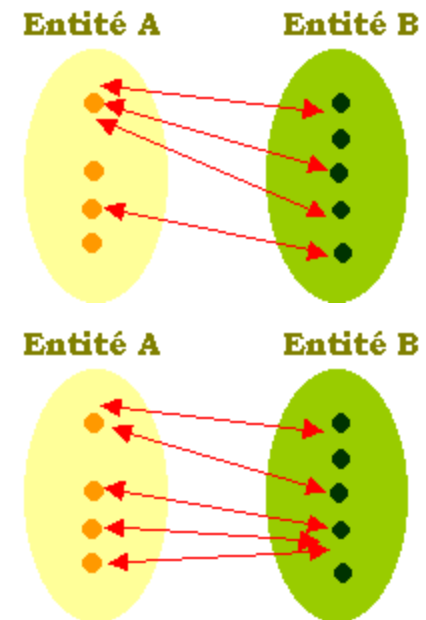
# RELATIONS OBLIGATOIRES

Notation E-A	Explication
1,1 <-> 1,1	<b>TOUTE</b> occurrence de A a un homologue <b>UNIQUE</b> parmi les occurrences de B et réciproquement
1,N <-> 1,N	<b>TOUTE</b> occurrence de A a <b>AU MOINS</b> un homologue parmi les occurrences de B et réciproquement



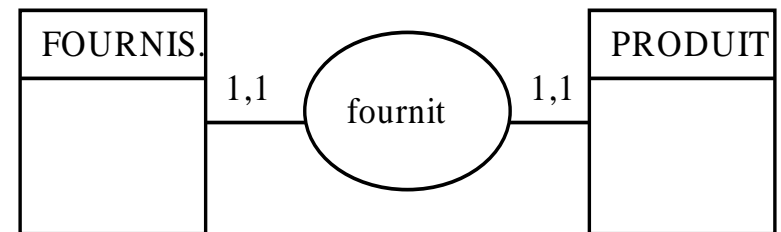
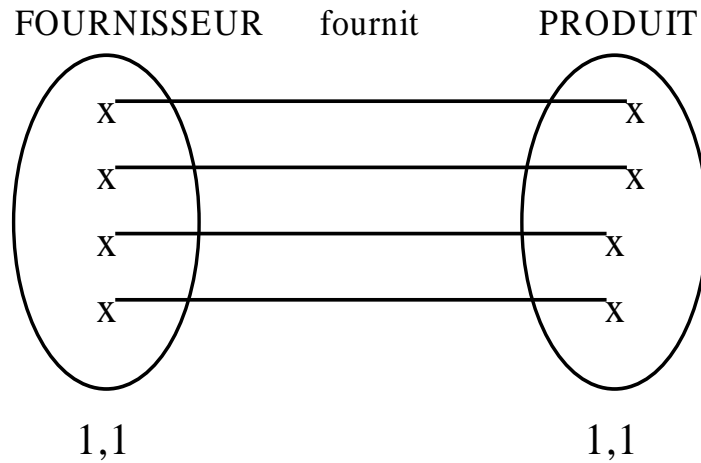
# RELATIONS OPTIONNELLES

Notation E-A	Explication
<b>0,N &lt;-&gt; 0,1</b>	UNE occurrence de A peut avoir 0,1,N vis-à-vis. UNE occurrence de B est limitée à 0 ou 1 homologue
<b>1,N &lt;-&gt; 0,N</b>	TOUTE occurrence de A a AU MOINS un homologue. Mais UNE occurrence de B peut ne pas en avoir, en avoir 1 ou plusieurs



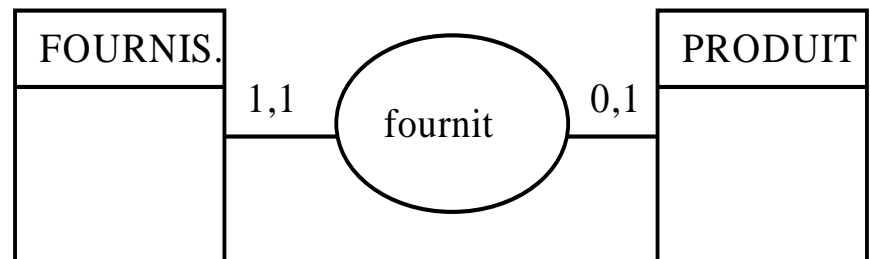
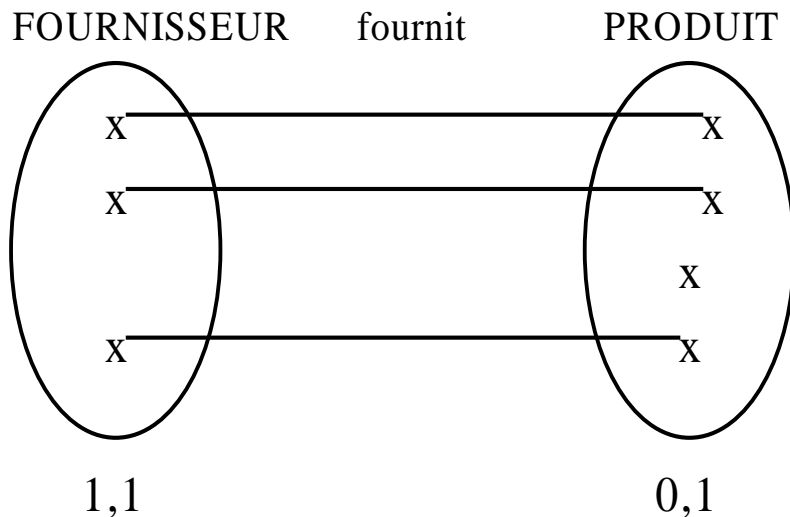
# LES ASSOCIATIONS DE TYPE 1-1

Tout fournisseur ne fournit qu'un seul produit.  
Tout produit n'est fourni que par un seul fournisseur.



# LES ASSOCIATIONS DE TYPE 1-1

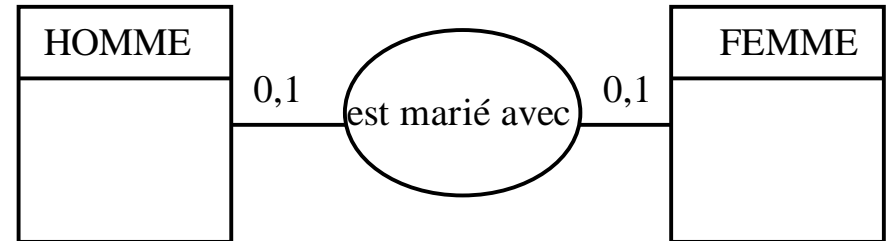
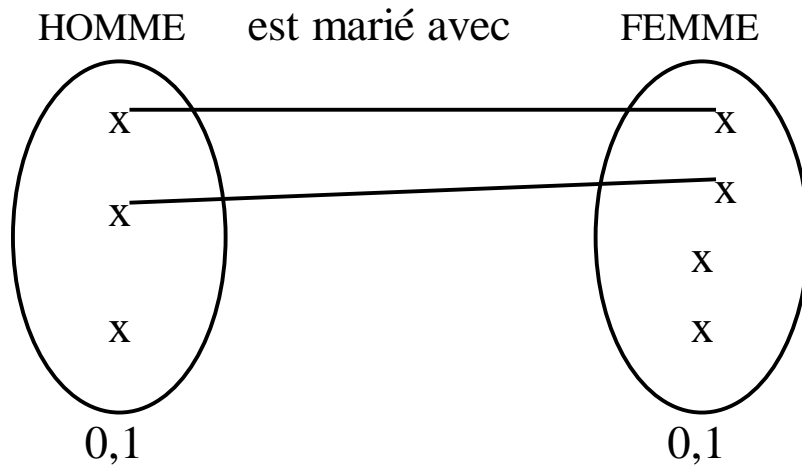
Certains produits sont fabriqués par l'entreprise et ne font donc pas l'objet d'approvisionnement.





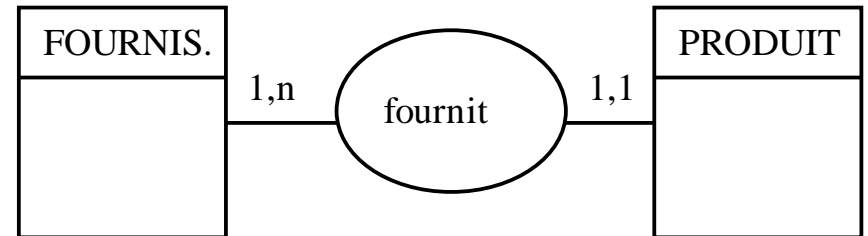
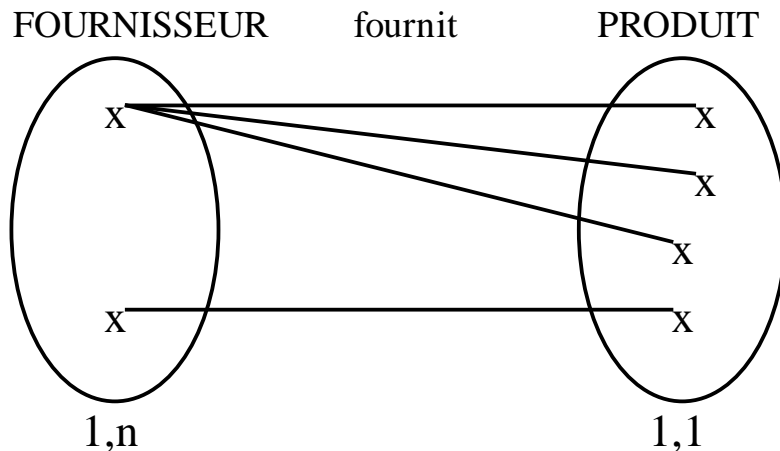
# *LES ASSOCIATIONS DE TYPE 1-1*

Un autre exemple ("être marié avec...") :



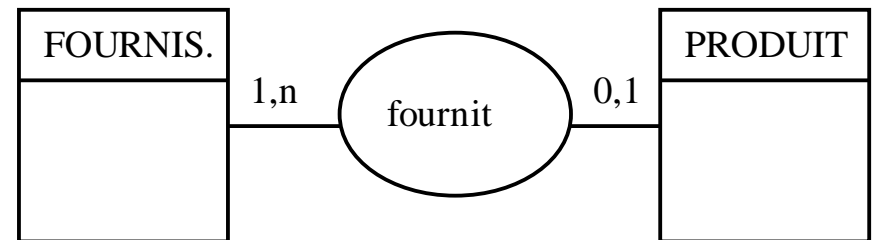
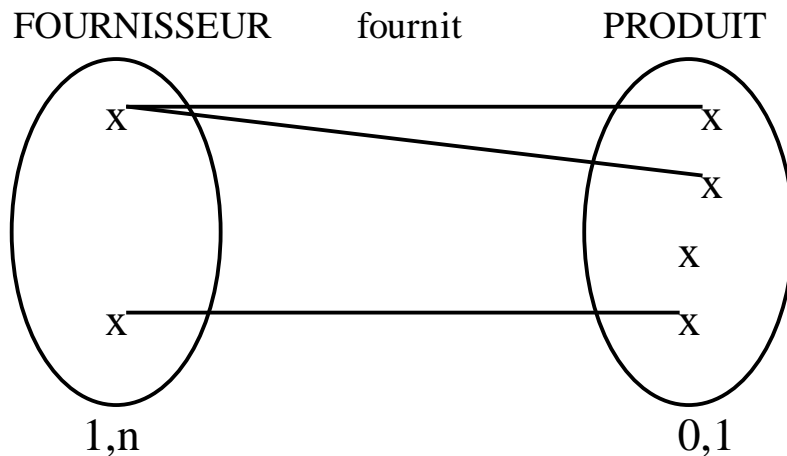
# *LES ASSOCIATIONS DE TYPE 1-N*

Tout fournisseur peut fournir un ou plusieurs produits.  
Tout produit n'est fourni que par un seul fournisseur.



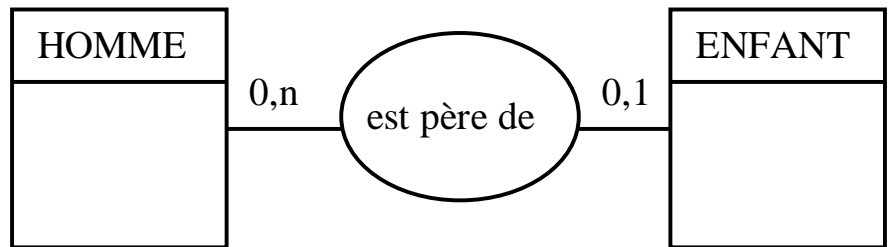
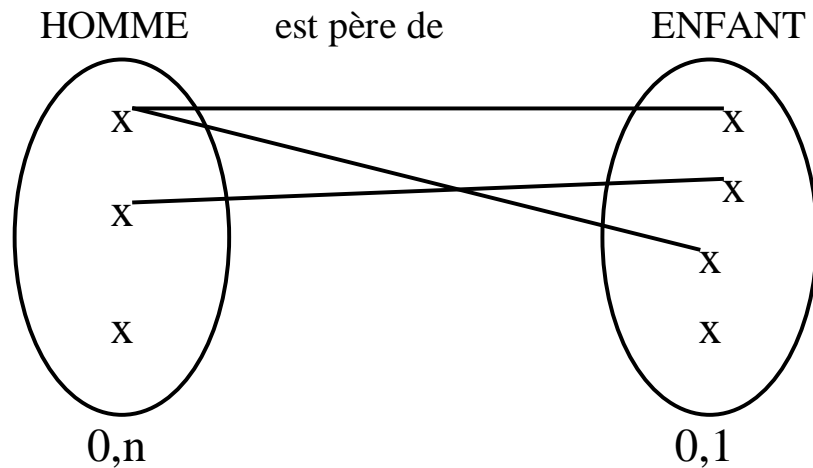
# LES ASSOCIATIONS DE TYPE 1-N

Certains produits sont fabriqués par l'entreprise et ne font donc pas l'objet d'approvisionnement.



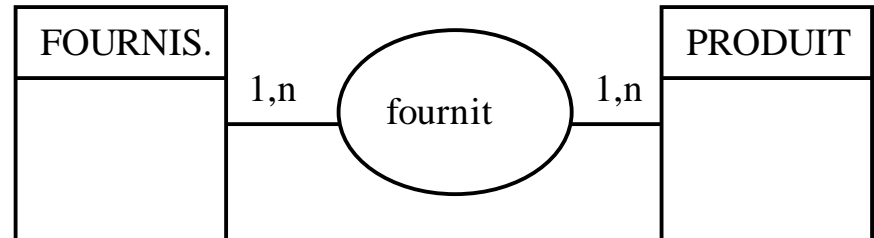
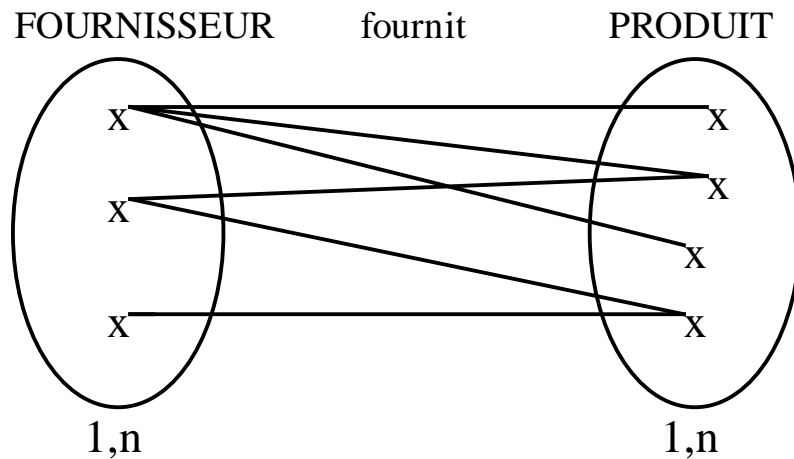
# *LES ASSOCIATIONS DE TYPE 1-N*

Un autre exemple ("être père de ...") :



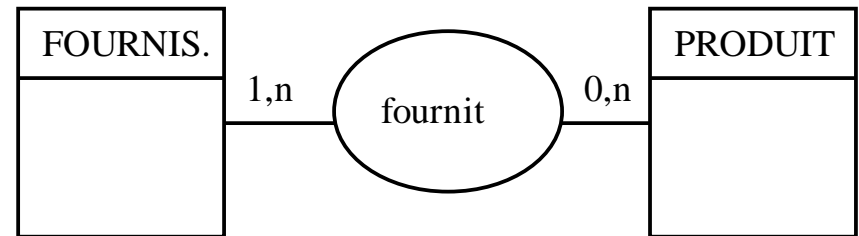
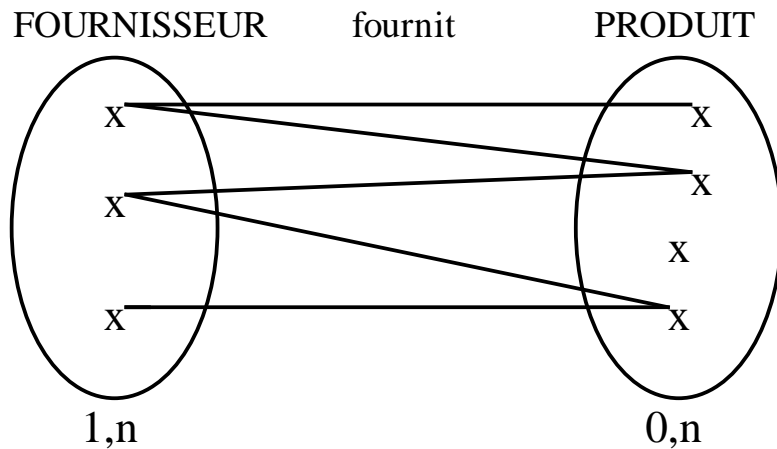
# *LES ASSOCIATIONS DE TYPE M-N*

Tout fournisseur fournit un ou plusieurs produits.  
Tout produit est fourni par un ou plusieurs fournisseurs.



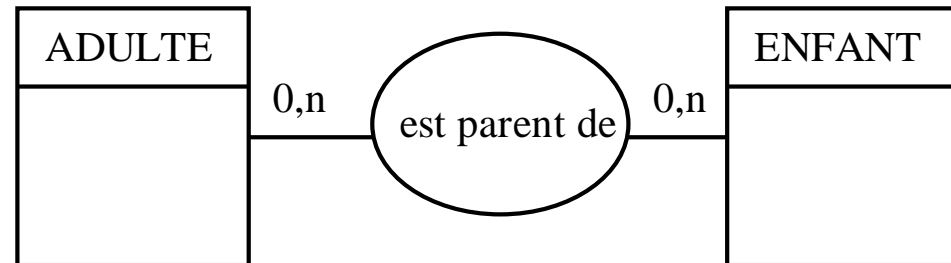
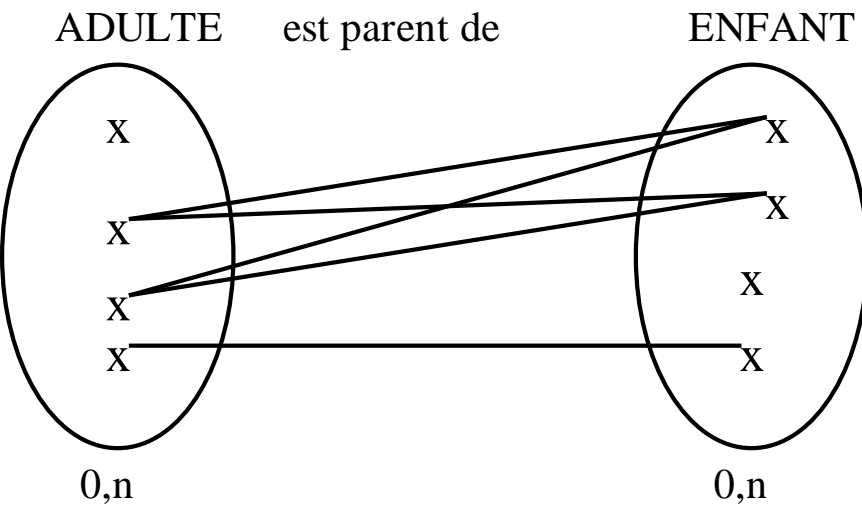
# LES ASSOCIATIONS DE TYPE M-N

Certains produits sont fabriqués par l'entreprise et ne font donc pas l'objet d'approvisionnement.



# *LES ASSOCIATIONS DE TYPE M-N*

Un autre exemple ("être parent de ...") :



## *CARDINALITÉ MINIMALES ÉGALE À 1*

- Une cardinalité minimale égale à 1 lie l'existence d'une occurrence de l'entité-type à sa participation à l'association.
- Un bon de livraison ne peut exister que s'il correspond à une commande.
- La création d'une occurrence de l'entité-type implique donc la création du lien correspondant. Ces deux actions seront réalisées au cours de la même opération



## *CARDINALITÉ MINIMALES ÉGALE À 0*

- Une cardinalité minimale égale à 0 rend indépendante l'existence d'une occurrence de l'entité-type de sa participation à l'association.
- Lorsqu'un nouveau produit est ajouté au catalogue, il n'a pas encore fait l'objet d'une commande.
- La création d'une occurrence de l'entité-type est déconnectée, dans le temps, de la création éventuelle du lien correspondant. Ces deux actions peuvent être réalisées par des opérations différentes

## *IDENTIFIANT D'ENTITE*

- Propriété PARTICULIERE de l'entité telle que pour chacune des valeurs de cette propriété, il existe une occurrence UNIQUE de l'entité.
- L'identifiant est inscrit en tête de la liste des propriétés et souligné.

Dans les modèles très denses il peut suffire à résumer les autres propriétés, pour faciliter la lecture.

## *IDENTIFIANT D'ASSOCIATION*

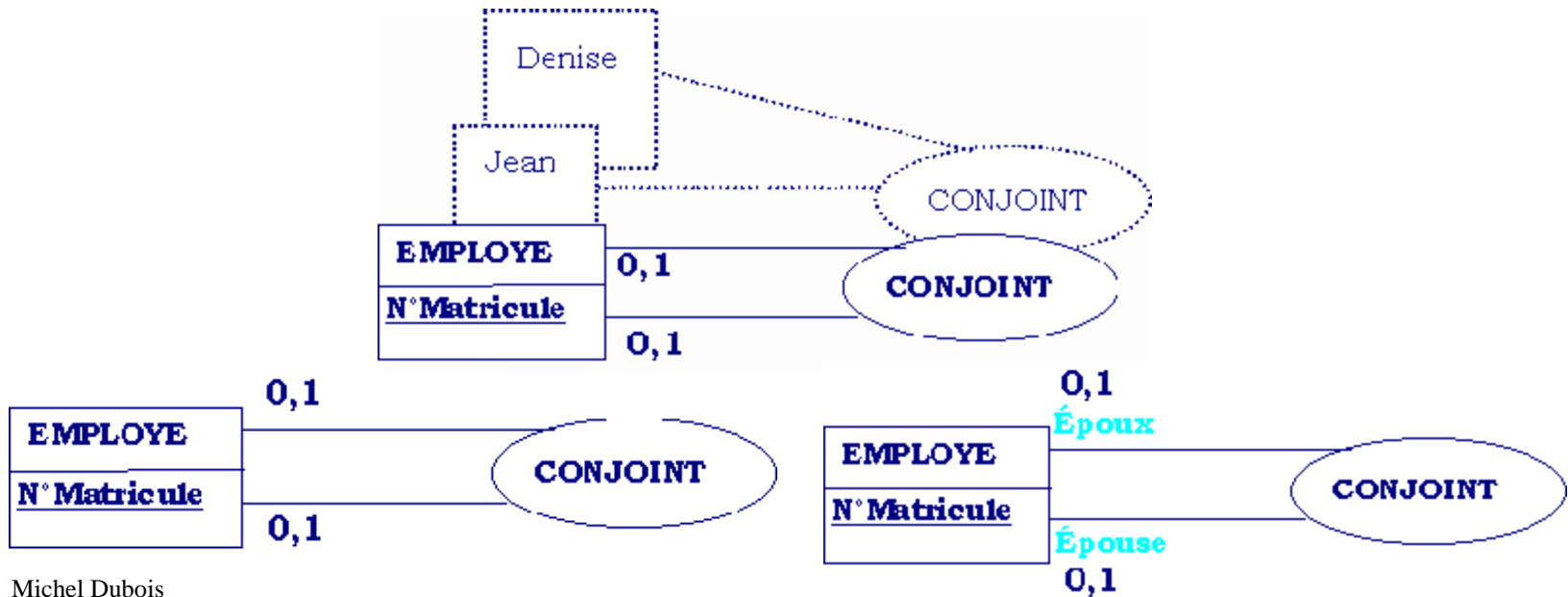
- Une association N'A PAS D'IDENTIFIANT explicite : l'association dépend des entités qu'elle relie.
- Son identifiant se déduit par calcul du produit cartésien des identifiants des entités associées.
- Pour l'association CONCERNE qui relie COMMANDE à PRODUIT, l'identifiant est le produit cartésien de N° Commande et N° Produit.

# *DIMENSIONS D'UNE ASSOCIATION*

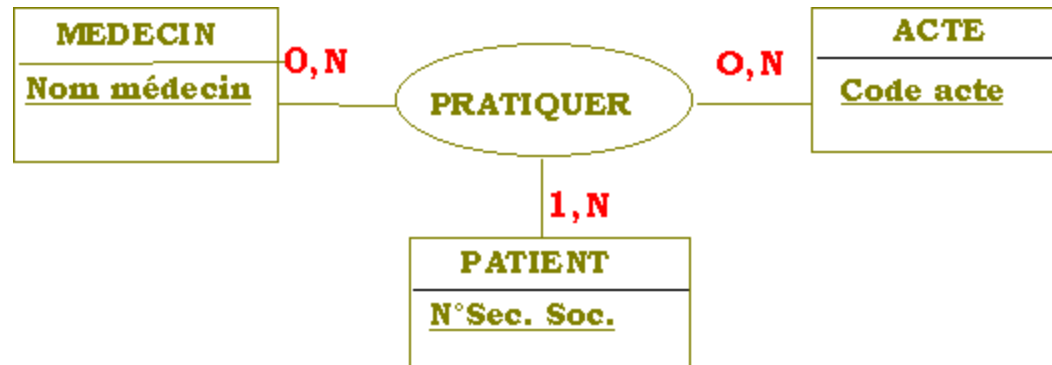
- On appelle DIMENSION d'une association le nombre d'entités qu'elle relie. Elle correspond à son nombre de "pattes".
- Une association qui relie seulement deux entités est donc un cas particulier.
- Il n'existe pas de limite au nombre de pattes d'une association. Cependant un nombre élevé de pattes est un indice que l'étude a été superficielle et approximative. (Il existe en effet un procédé de simplification)

# ASSOCIATION REFLEXIVE

- Une association "réflexive" est une association qui lie des occurrences d'une même entité entre elles.
- C'est un cas particulier de la dimension 2.
- Pour éviter toute ambiguïté, il vaut mieux indiquer le RÔLE joué distinctement par chacune des occurrences.

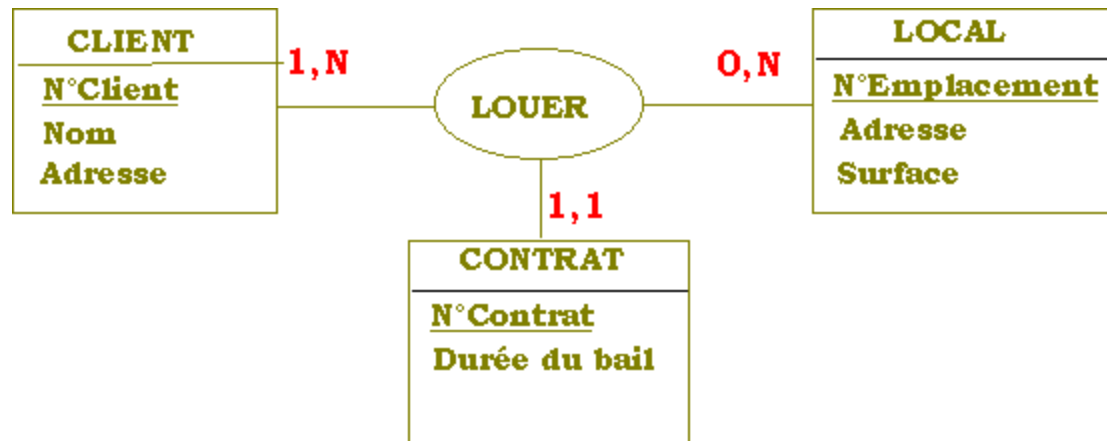


# *DIMENSION SUPÉRIEURE À 2 D'UNE ASSOCIATION ET CARDINALITÉS*



- **Pour une occurrence de médecin, il peut ne pas y avoir de couple patient \* acte** (le Directeur du centre par exemple qui ne participe pas aux consultations) **mais en général il y en a plusieurs.**
- **Pour un acte il peut ne pas y avoir de couple médecin \* patient** (un acte chirurgical qui n'est pas pratiqué dans notre centre par exemple) **mais éventuellement plusieurs.**
- **Pour un patient il y a au moins un couple médecin \* acte et éventuellement plusieurs.**

# ASSOCIATION N-AIRE INVALIDE



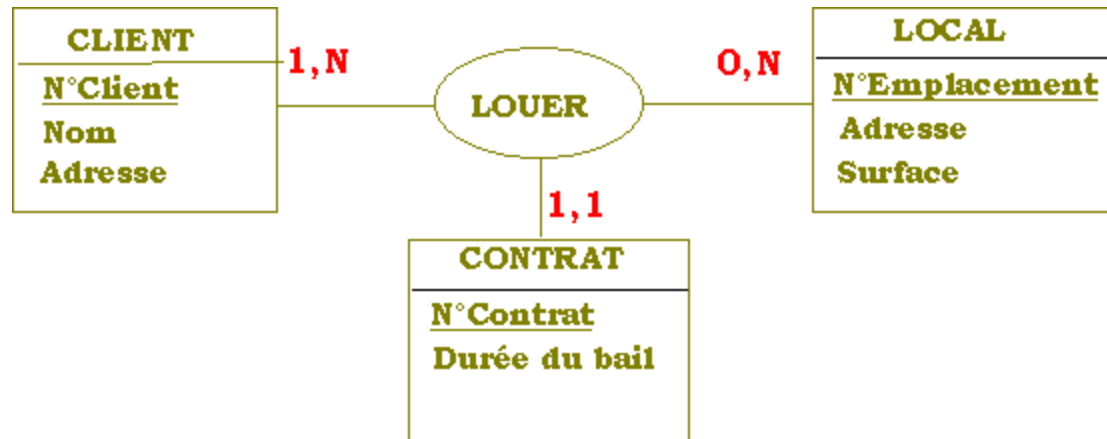
Pour une occurrence de client, il y a au moins un couple local \* contrat et au plus plusieurs

Pour un local il peut ne pas y avoir de couple client \* contrat mais éventuellement plusieurs

Pour un contrat il y a 1 et 1 seul couple client \* local

- si l'on trouve une cardinalité 1,1 sur l'une des pattes, le modèle peut être décomposé

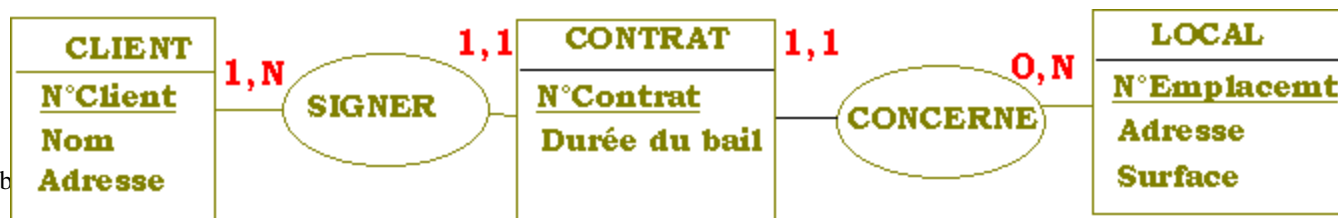
# DÉCOMPOSITION SYSTÉMATIQUE D 'UNE ASSOCIATION N-AIRE



**l'association LOUER est indirectement identifiée par N° Client \* N°Emplacement \* N° Contrat.**

**N° Contrat -> N° Client, N°Emplacement**

**Pour identifier une occurrence de LOUER parmi toutes les autres, N° Contrat suffit.**





# *CONTRAİNTE D'INTEGRITE FONCTIONNELLE*

- **Une Contrainte d'Intégrité Fonctionnelle (en abrégé : CIF) se définit par le fait que l'une des entités participant à l'association est complètement déterminée par la connaissance d'une ou plusieurs autres entités participant dans cette même association.**
- **Cas des CIF pour une association de dimension 2**
- **Les CIF permettent de décomposer les relations de dimensions supérieures à 2.**

# *CIF ET ASSOCIATION DE DIMENSION 2*

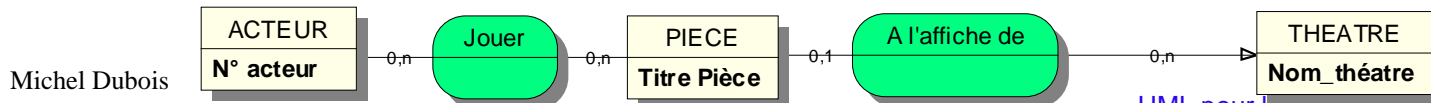
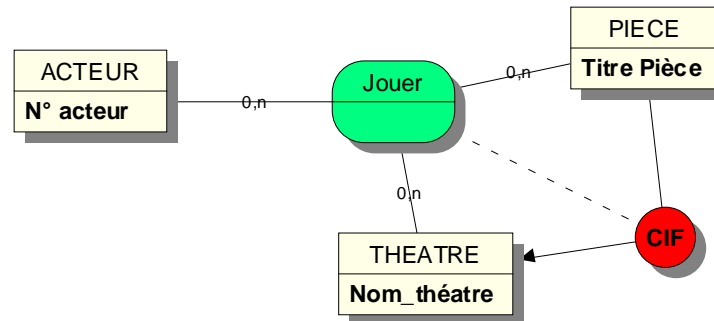
- **La CIF consiste simplement en une cardinalité 1,1 sur l'une des pattes.**
- **Les associations qui comportent une cardinalité 1,1 sur l'une des pattes ne sont jamais porteuses de propriétés. Celles-ci "migrent" sur l'entité déterminante.**



# *CIF ET ASSOCIATION DE DIMENSION SUPÉRIEURE À 2*

- **lorsqu'il y a une CIF,**
- **lorsque la dépendance fonctionnelle concerne un nombre d'entité inférieur à la dimension de la relation**
- **Alors l'entité déterminée peut être détachée de l'association initiale pour rester associée avec la seule entité déterminante.**

les théâtres ont l'exclusivité  
des pièces représentées :

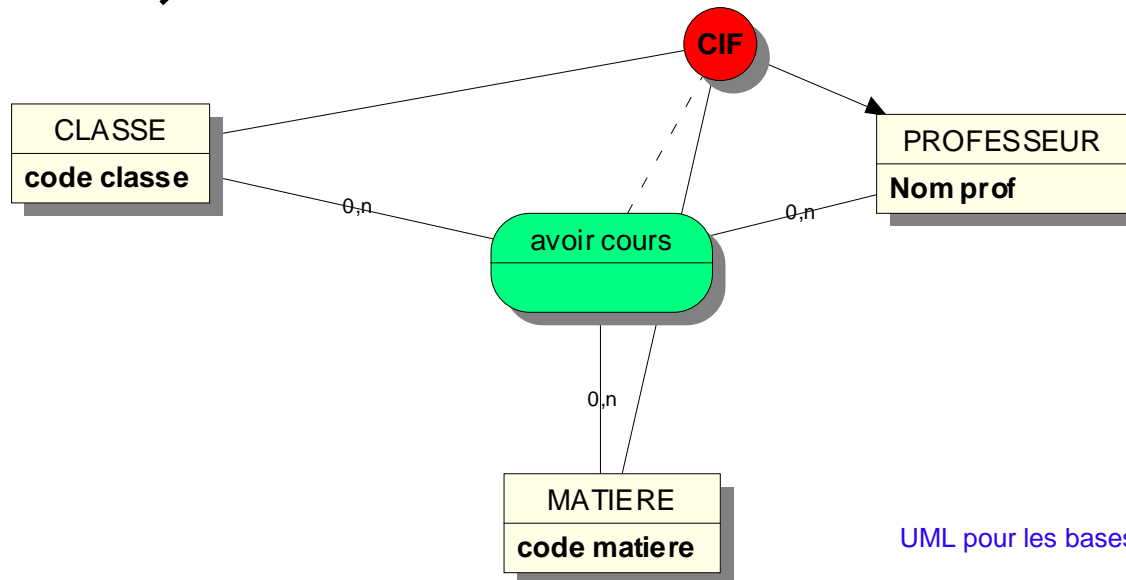


# CIF ET ASSOCIATION NON DÉCOMPOSABLE

***Dans un lycée, les professeurs donnent des cours à des classes dans certaines matières.***

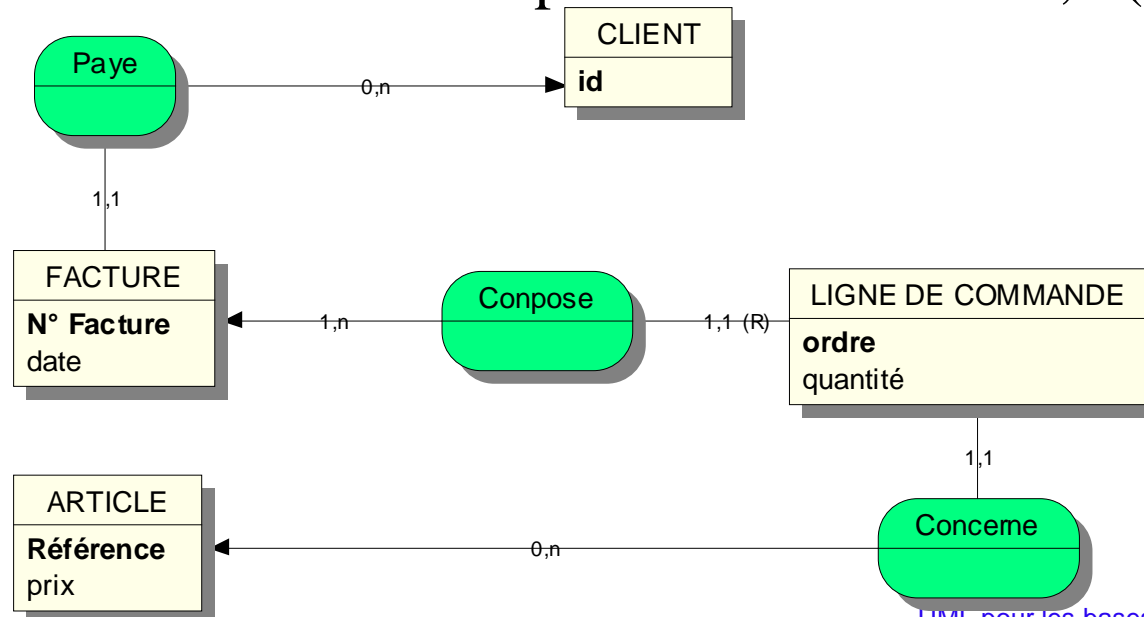
***Mais on sait encore que dans une matière, une classe n'a qu'un seul professeur.***

**CLASSE , MATIERE -> PROFESSEUR**



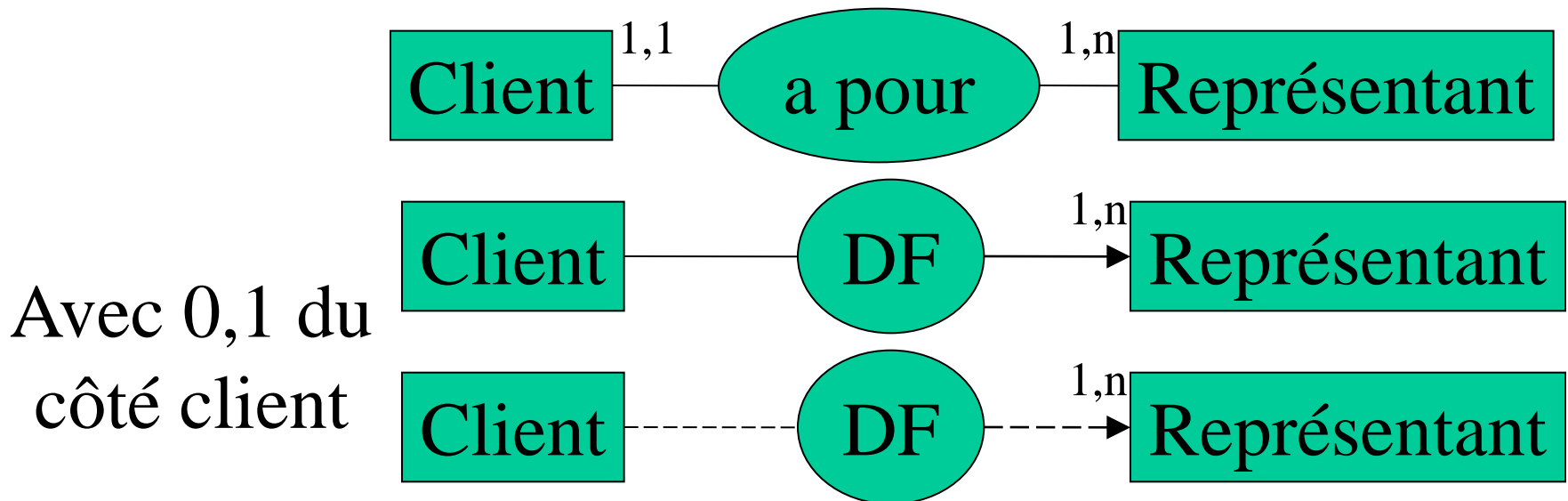
# *IDENTIFICATION RELATIVE*

- Certaines entité type ont par ailleurs une existence totalement dépendante d 'autres entité type. Il s 'agit d 'entité faible.
- L 'identification relative s 'effectue par par une propriété stable de cette entité qui ne remplit pas les conditions d 'un identifiant absolu et par une cardinalité 1,1 (R).



# Les dépendances fonctionnelles entre entités

- Entre 2 entités du MCD, il existe une dépendance fonctionnelle (DF) forte si à une occurrence de l'entité source est associée une et une seule occurrence de l'entité cible
- Cette dépendance est dite faible si la cardinalité minimale est 0
- Exemple :

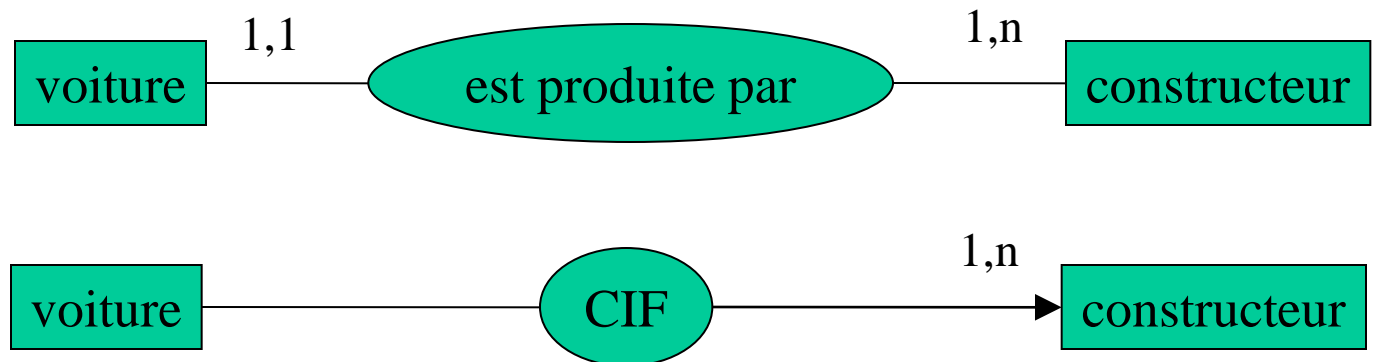


# Les contraintes d'intégrité

- Elles reflètent ou respectent les règles de gestion de l'entreprise
- Les cardinalités sont un cas de contraintes d'intégrité
- Les contraintes d'intégrité syntaxique portent sur une propriété et peuvent concerner soit sa forme soit son domaine de valeur
  - Un prix doit être un réel strictement positif
  - Une date prend la forme jj/mm/aaaa
- Les contraintes d'intégrité sémantique s'appliquent à plusieurs propriétés ensemble. Ces propriétés ne sont pas forcément des éléments d'une seule entité ou d'une seule association
  - Heure de départ < heure d'arrivée

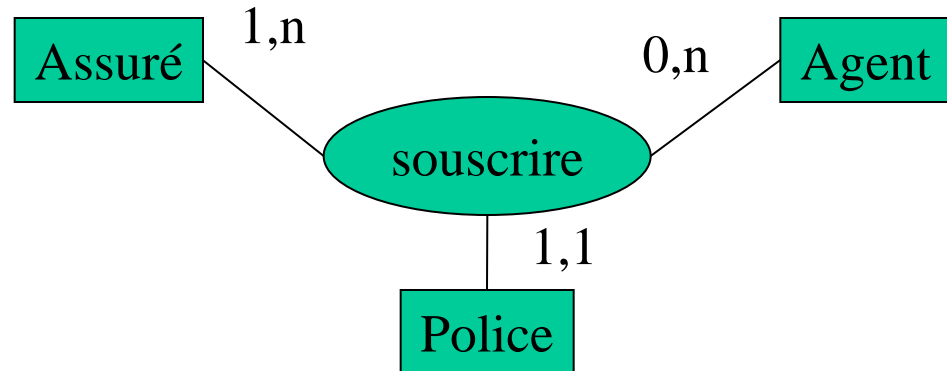
# Les contraintes d'intégrité

- Une contrainte d'intégrité fonctionnelle (CIF) ou DF totale est un cas particulier de la DF forte :
  - Le lien existe de manière unique pour toutes les occurrences de la source
  - La dépendance est stable, ie une fois le lien établi entre 2 occurrences, il ne peut être modifié
- Exemple :





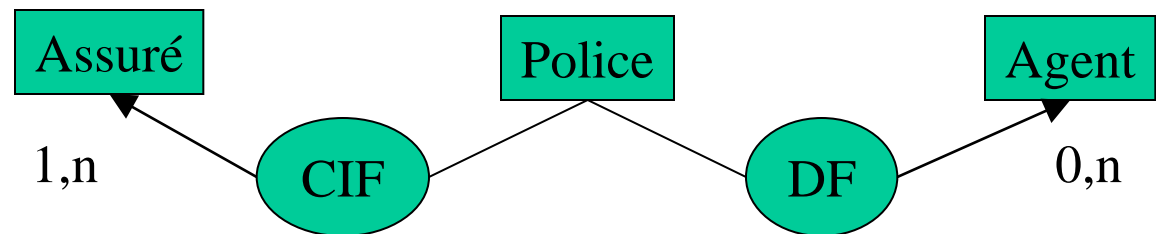
# CIF et DF pour simplifier le MCD



- Modèle initial

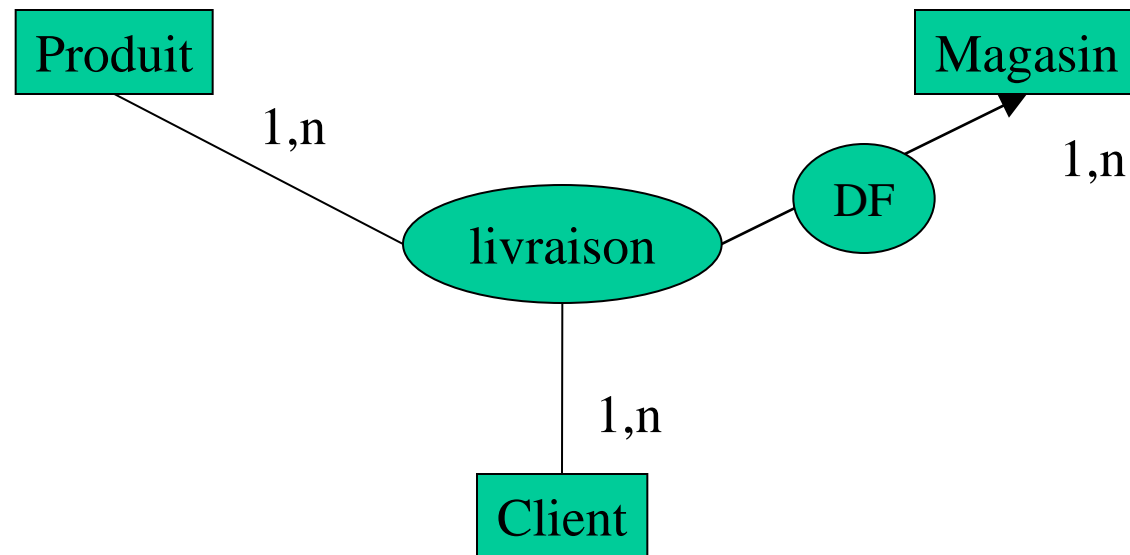
- RG1 : une police n'est souscrite que par un et un seul assuré et elle n'est pas cessible : on a une CIF entre police et assuré
- RG2 : Une police est gérée par un et un seul agent mais il est possible qu'il change au cours du temps : on a une DF forte entre police et agent

- D'où :



# CIF et DF pour simplifier le MCD

- La source d'une dépendance fonctionnelle peut être une relation



- On a :  
 $\text{Client} \times \text{Produit} \rightarrow \text{Magasin}$

# DF non triviales et MCD

- Tout identifiant d'une entité détermine fonctionnellement toutes les autres propriétés de cette entité
- Toute cardinalité maximale à 1 implique une DF dont la source est l'identifiant de l'entité concernée par cette cardinalité et la cible les identifiants des entités associées
- Toute association porteuse de propriétés implique une DF dont la source est composée des identifiants des entités associées et la cible l'ensemble des propriétés portées par l'association

# Règles concernant les entités

- Une entité possède toujours au moins une propriété : son identifiant
- Chacune des propriétés d'une entité doit caractériser toute occurrence de cette entité de la même manière
- S'il existe une occurrence d'association alors il existe nécessairement une occurrence de chacune des entités associées
- Deux occurrences d'une entité ne peuvent participer à la même occurrence de l'association (sauf si l'association a deux branches vers cette entité)

# Règles concernant les propriétés

- Une même propriété ne peut figurer que sur un seul objet (entité ou association)
- Une propriété doit être élémentaire (atomique) de telle sorte qu 'on ne puisse pas la décomposer (1NF)
- Une propriété doit dépendre pleinement de son identifiant et pas seulement d 'une partie de celui-ci (2NF)
- Une propriété doit dépendre directement de l 'identifiant (sans passer par l 'intermédiaire d 'une autre propriété) (3NF)

# Extension du modèle

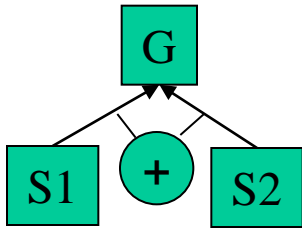
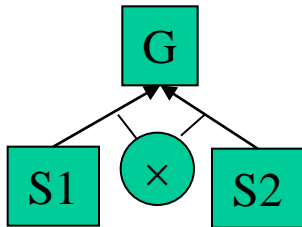
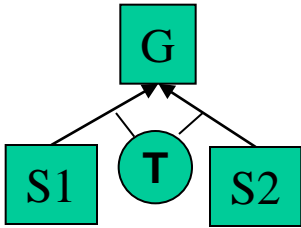
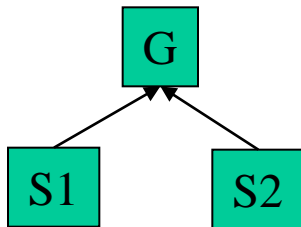
- Existence de valeurs nulles ou insignifiantes (card min = 0)
- Redondance des propriétés
- Description pas assez fine des relations entre objets
- Sémantique insuffisamment prise en compte
- Pour pallier ces inconvénients : introduction du concept de hiérarchie de généralisation/spécialisation
- Un exemple :



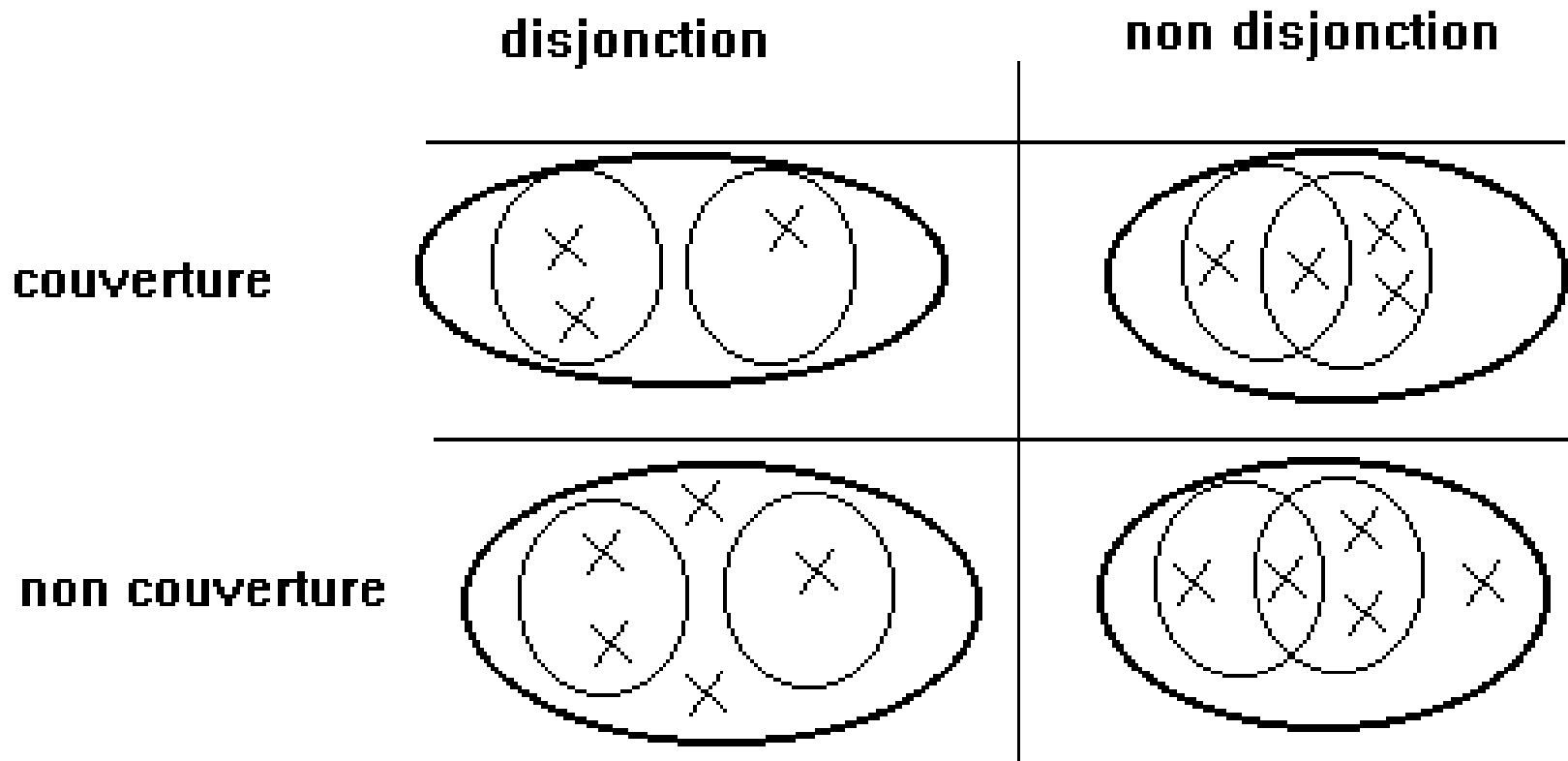
- 3 types d'employés : secrétaires, ingénieurs et chauffeurs
- 2 sortes de véhicules : voiture et camion

# Extension du modèle

- Couverture : toute occ. du type générique doit appartenir à l'une des spécialisations
- Disjonction : une occ. d'un sous-type ne peut appartenir qu'à un sous-type

	Couverture	Non couverture
Disjonction		
Non disjonction		

# *CONSTRAINTES DE SPÉCIALISATION*



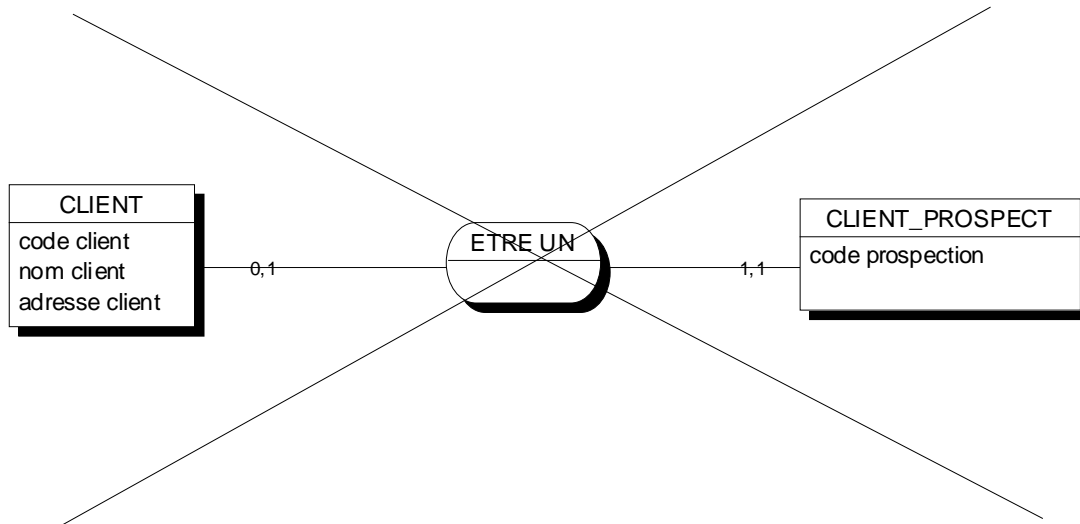


# Extension du modèle

- L'entité spécialisée hérite :
  - Des propriétés de l'entité générique
  - Des relations associées à l'entité générique
- Identifiant : il peut y avoir plusieurs identifiants correspondant à des vues externes différentes
  - Identifiant total : permet d'identifier chacune des occurrences d'un type
  - Identifiant relatif : intègre une relation stable (CIF)
$$\text{id source} = \text{id cible} + \text{numéro d'ordre}$$

# *GÉNÉRALISATION-SPÉCIALISATION*

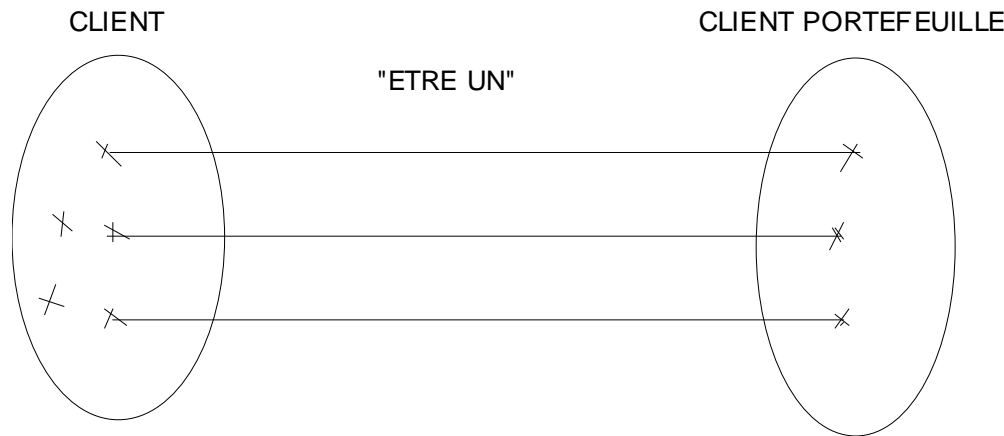
Pour représenter la généralisation-spécialisation, quelques ouvrages proposent le schéma suivant :



# GÉNÉRALISATION-SPÉCIALISATION

Cette représentation est dangereuse à plus d'un titre. En effet, elle laisse supposer :

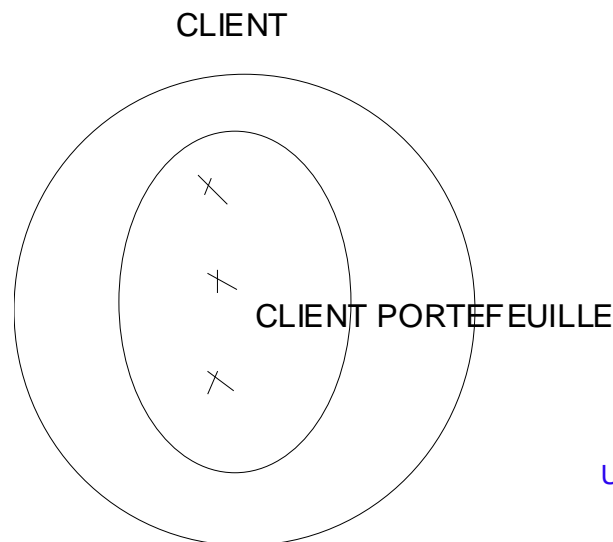
- que la généralisation-spécialisation peut être représentée par une simple relation entre ensembles (ce qui donne un schéma pour le moins étonnant !).



- qu'il existe un lien entre deux occurrences d'entités-type (celles-ci ayant chacune son existence propre) ; cela ne correspond évidemment pas à la réalité ! La relation de généralisation-spécialisation permet de considérer le même client avec plus ou moins de détails ; elle ne concerne donc qu'une et une seule occurrence d'entité-type.

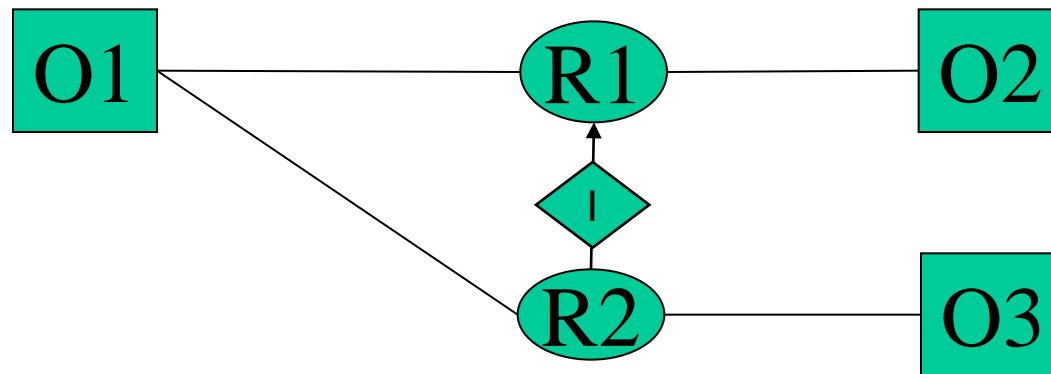
# *LIEN D'INCLUSION*

En conclusion, le concept de généralisation est bien une extension au modèle entité-association. Il exprime une relation d'inclusion entre un sous-type et sa classe générique. La généralisation-spécialisation permet de voir une même occurrence d'entité-type à différents niveaux d'abstraction.



# Extension du modèle

- Représentation des contraintes inter-relations :
  - Inclusion ou implication : les occ de la relation source sont incluses dans l'ensemble des occ de la relation cible





- Exclusion : la participation d'une occ de l'objet à une occ de relation exclut sa participation aux occ des autres relations liées par la contrainte

Représentation :  sur un lien non fléché

# Extension du modèle

- Totalité : toute occ de l'objet pivot participe au moins à une occurrence d'un des types de relations liées par le contrainte

Représentation :  lien supplémentaire sur l'objet pivot

- Partition : même chose en remplaçant au moins par  exactement
- Egalité ou simultanéité : égalité entre l'ensemble des occurrences des types de relations liées par la contrainte 
- Propriété multivaluée
  - Ensemble : sans doublon
  - Liste : avec doublon
- Surcharge des domaines de valeurs possible pour les sous-types (Personne.âge [0:120] ; Employé.âge [18:65])



# Université de Bretagne Sud

*Informatique*



## Génie Logiciel avec UML

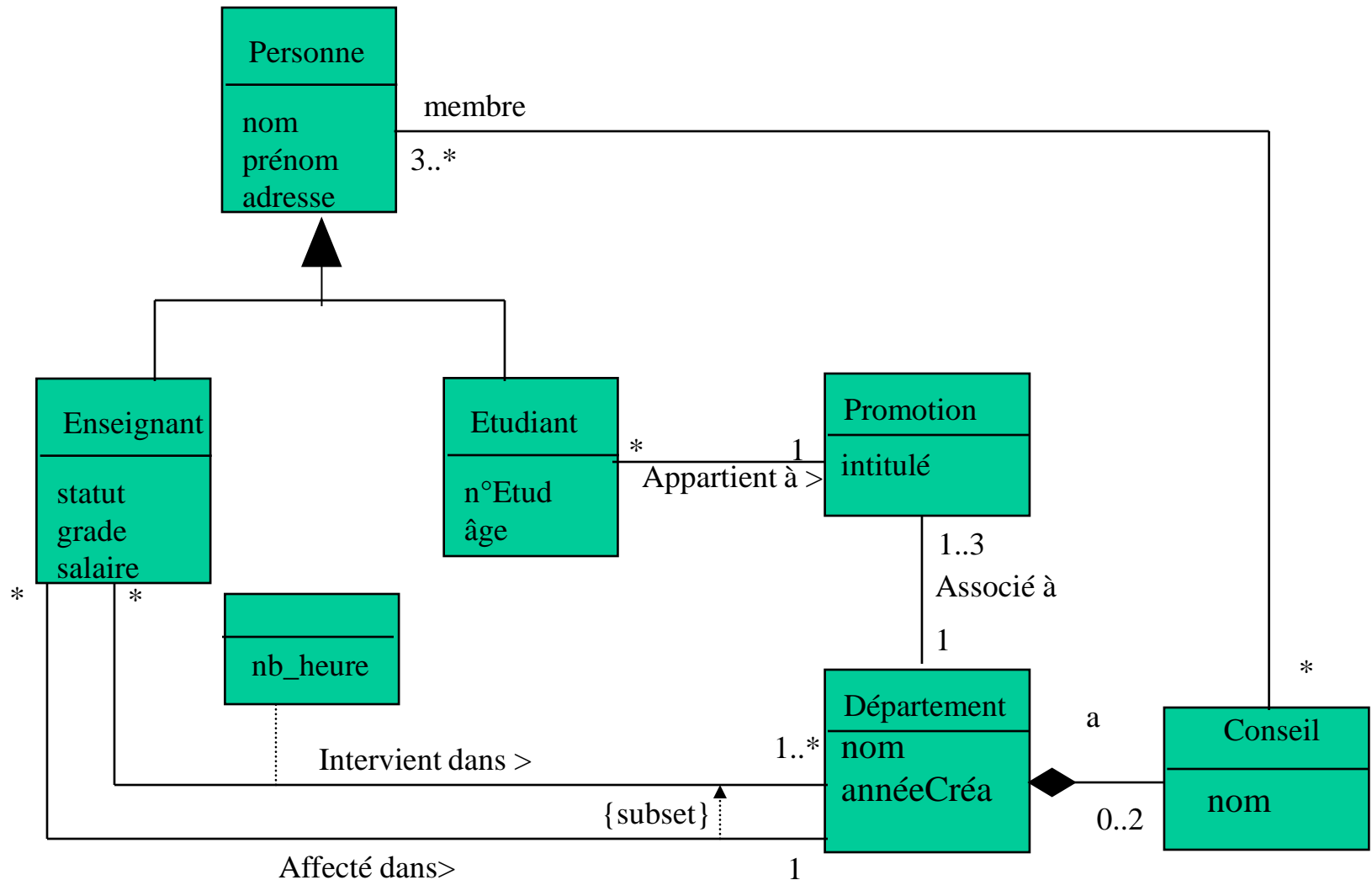


### Chapitre 4

## Introduction aux notations UML

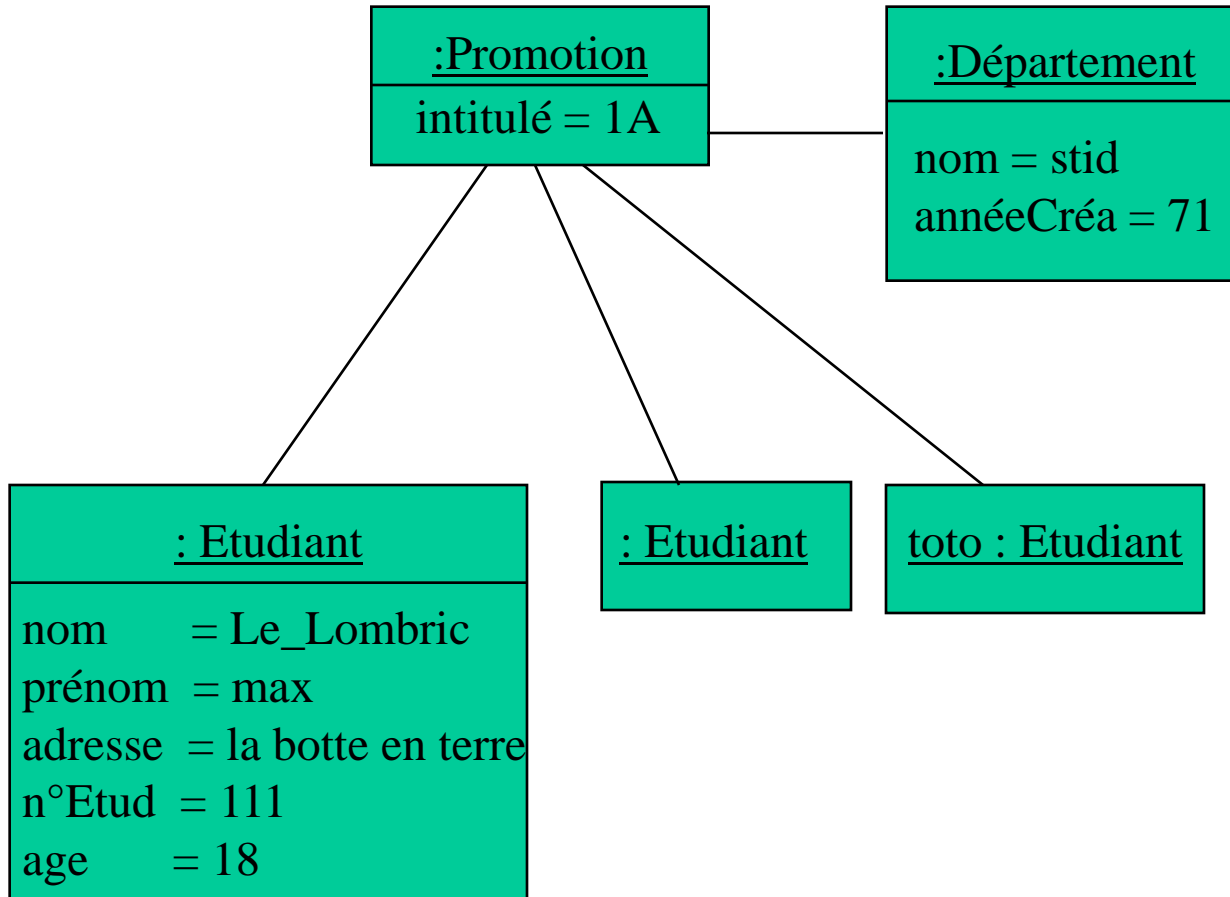
[Retour au plan](#)

# Le Diagramme De Classes





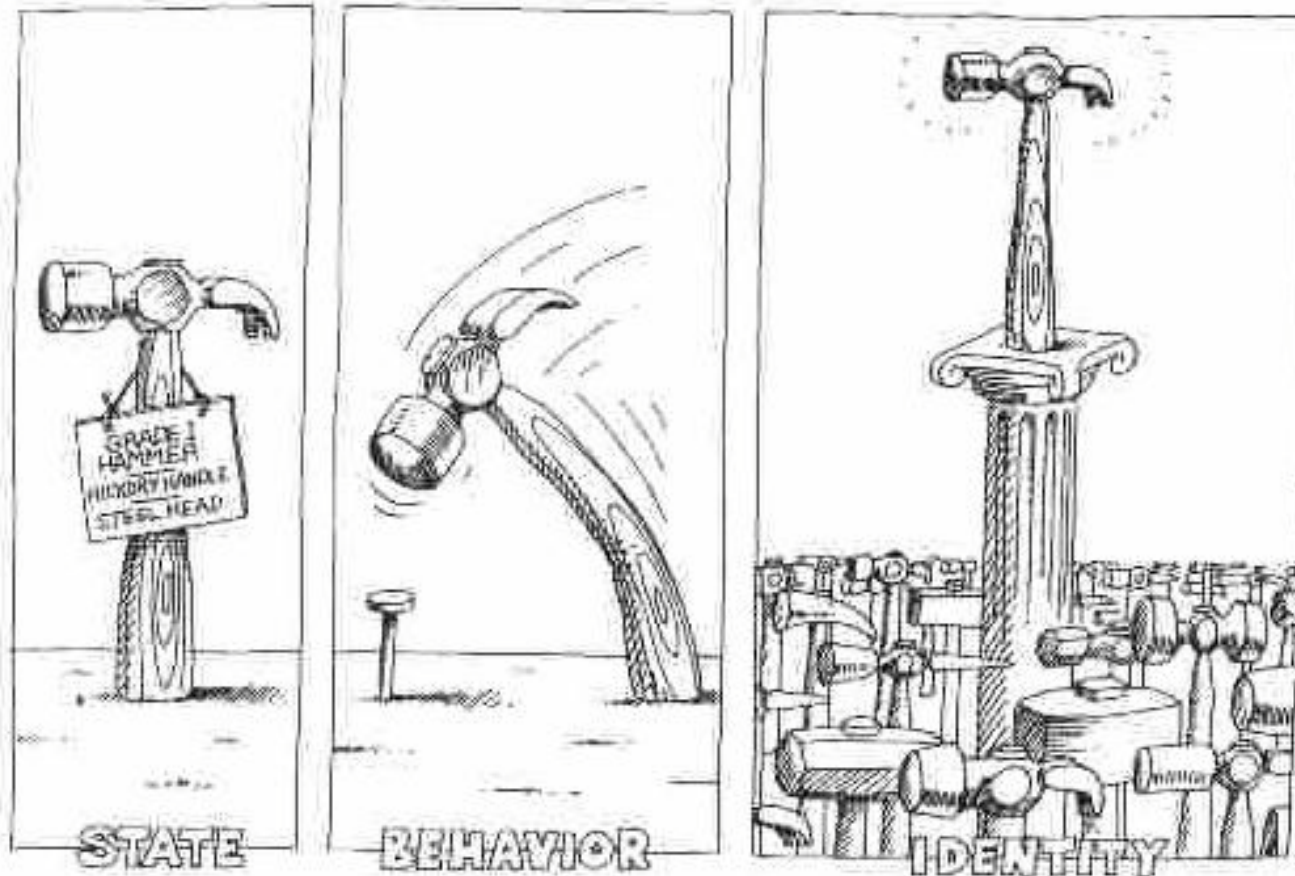
# Le Diagramme D'objets



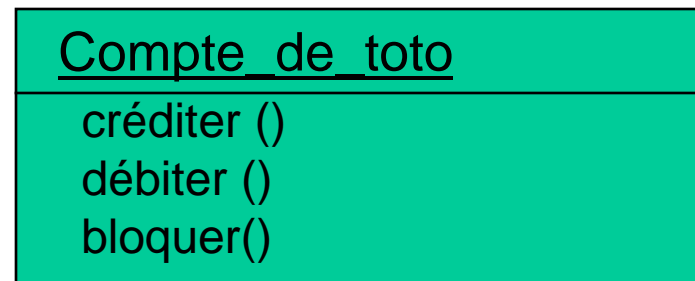
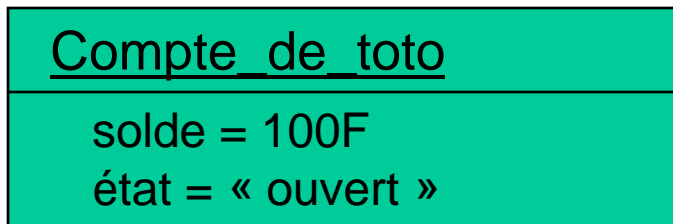
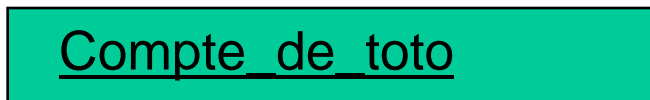
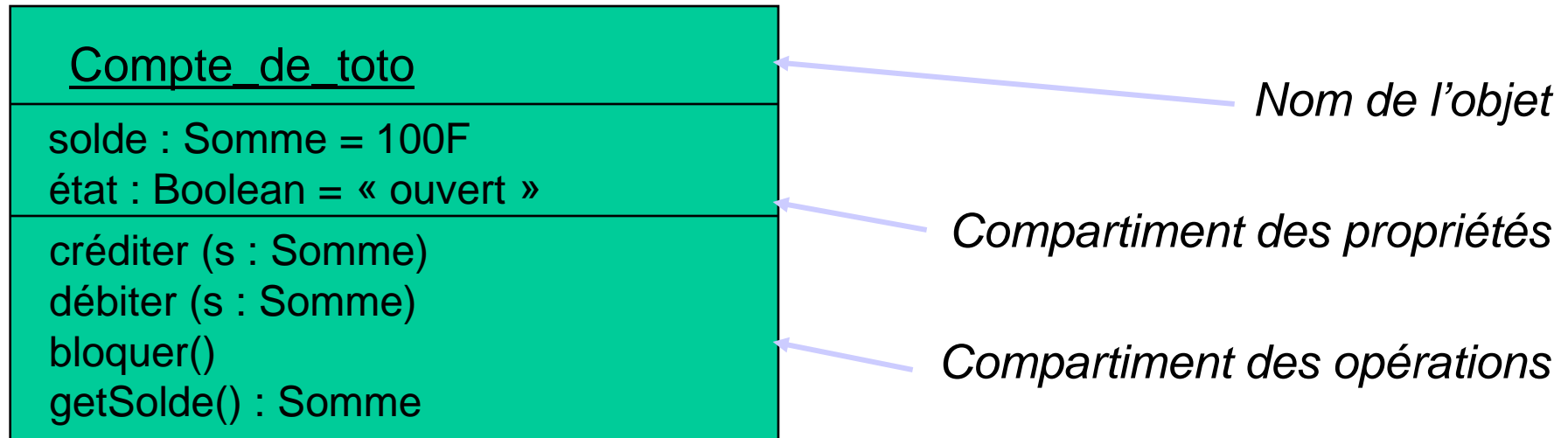
# Notion d'objet

- Dans le monde réel, notre esprit structure notre environnement selon le concept d'objet (chaise, table, etc.). La modélisation objet des logiciels rejoint la façon que nous avons nous même d'évoquer le monde qui nous entoure.
- Les objets de modélisation vont définir une réalité simplifiée (on parle d'abstraction) des entités du monde réel. Ces objets de modélisation deviendront au fil du développement des objets informatiques (au sens de la POO ou des BDOO) ou des tuples (au sens des SGBDR).
- Les objets de modélisation représente le plus souvent une abstraction d'entités massiques (ex : un client de banque Mr. James, l'avion 218 pour Honolulu) ou abstraites (ex : la trajectoire de l'avion pour Tokyo) du monde réel.

Comme pour les objets du monde réel, les objets de modélisation se voient associer un *état*, une *identité* et un *comportement* qui

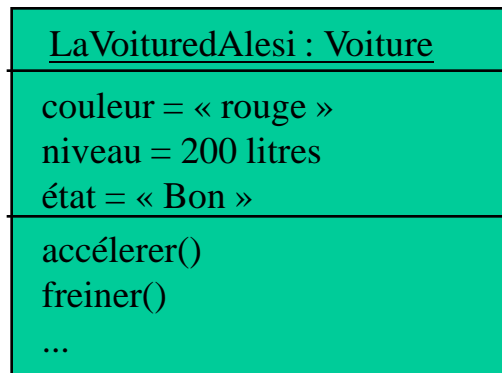
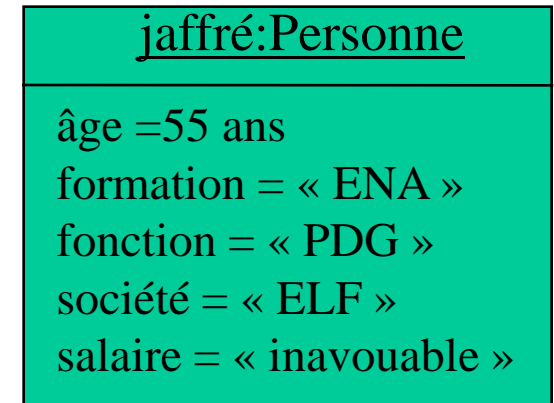


# Représentation UML d'un objet

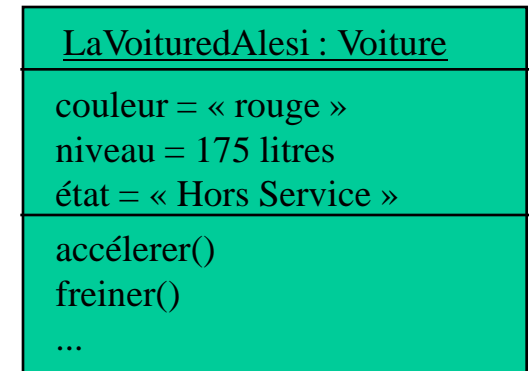


# L'état d'un objet

- L'état regroupe les valeurs instantanées de tous ses *attributs* et des liens qu'il entretient.
- Un attribut est une information particulière sur l'objet du monde réel modélisé.
- Chaque attribut peut prendre une valeur dans un domaine de définition donné. Un attribut est une variable typée.
- Cet état évolue au cours du temps



Deux tours  
plus tard



# Le comportement d'un objet

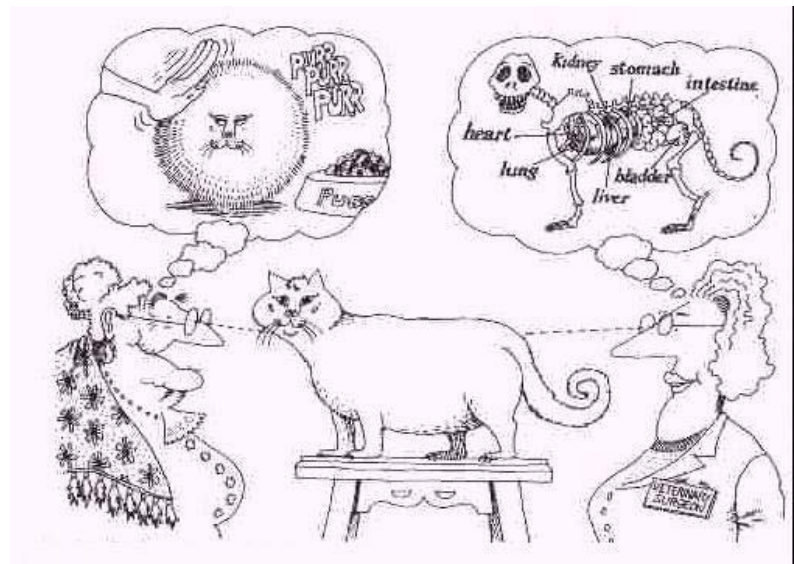
- Le comportement regroupe toutes les compétences d'un objet.
- Il se présente sous la forme d'opérations.
- Une opération représente un service offert par l'objet aux autres objets.
- Les opérations d'un objet sont déclenchées suite à une stimulation externe : l'envoi par l'objet client d'un message à l'objet fournisseur du service.
- Le message est dénoté par le nom de l'opération offerte par l'objet fournisseur que l'on souhaite stimuler.

# L'identité d'un objet

- Tous les objets du monde réel sont uniques. De la même façon, tout objet de modélisation ou d'implantation possède une identité qui lui est propre. Cet identité s'appelle OID pour Object Identifier.
- L'OID permet de distinguer tout objet de façon non ambiguë, indépendamment de son état.
- Il est nécessaire de connaître l'OID d'un objet pour accéder à ses services.
- Attention : 2 objets ayant des identités différentes mais ayant le même état sont différents!
- La notion d'identité est un concept qui ne se représente pas de manière explicite dans une modélisation objet.
- Dans les langages à objets cet identifiant est la référence vers l'objet. Dans une table de SGBDR c'est la valeur de clé primaire du tuple représentant l'objet.

# Un objet est une abstraction

- Une abstraction est un résumé, un condensé
- C'est un des moyens utilisés par l'homme pour gérer la complexité du monde qui l'entoure.
- Elle vise à dégager les seules caractéristiques essentielles au problème posé en ignorant les détails sans intérêt.
- Une abstraction se définit par rapport à un point de vue.

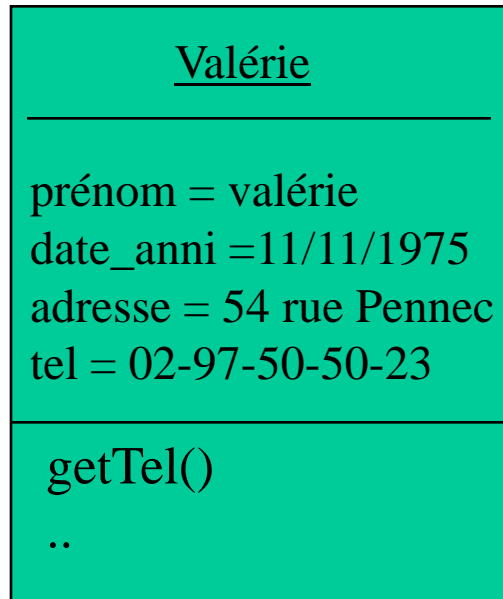




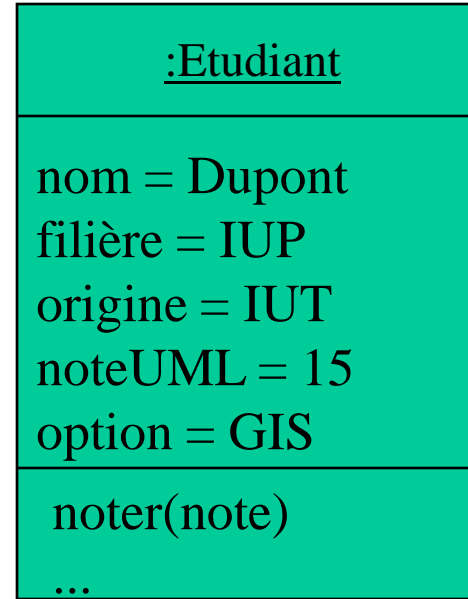
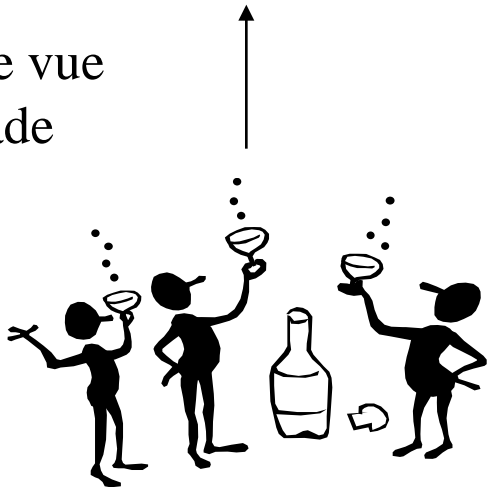
# Notion de classe et d'instance

- Le monde qui nous entoure est constitué de très nombreux objets. Pour le comprendre, l'être humain a tendance à regrouper les éléments qui se ressemblent.
- Regrouper des objets suivant des critères de ressemblance s'appelle classer.
- Les biologistes ont ainsi classé les animaux en différentes espèces : vipères, moustiques, mouches, etc.
- Ce concept de classement a été repris en modélisation objet. On factorise la description d'objets présentant les mêmes propriétés dans une classe.
- Une classe représente donc un moule à partir duquel il sera possible de créer des objets qui présenteront tous les mêmes types d'attributs et les mêmes opérations
- Le processus permettant d'obtenir un objet depuis une classe se nomme instantiation. On parle indifféremment d'objet ou d'instance.

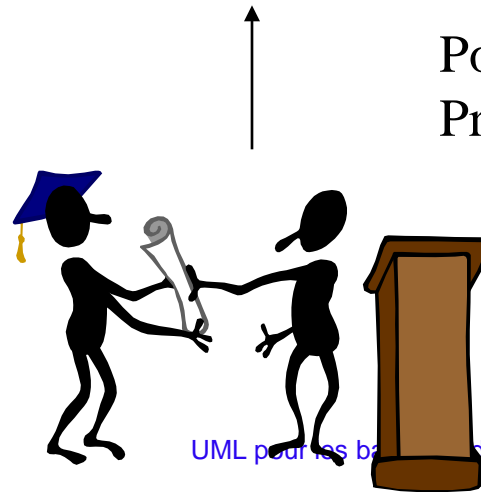
# Un exemple



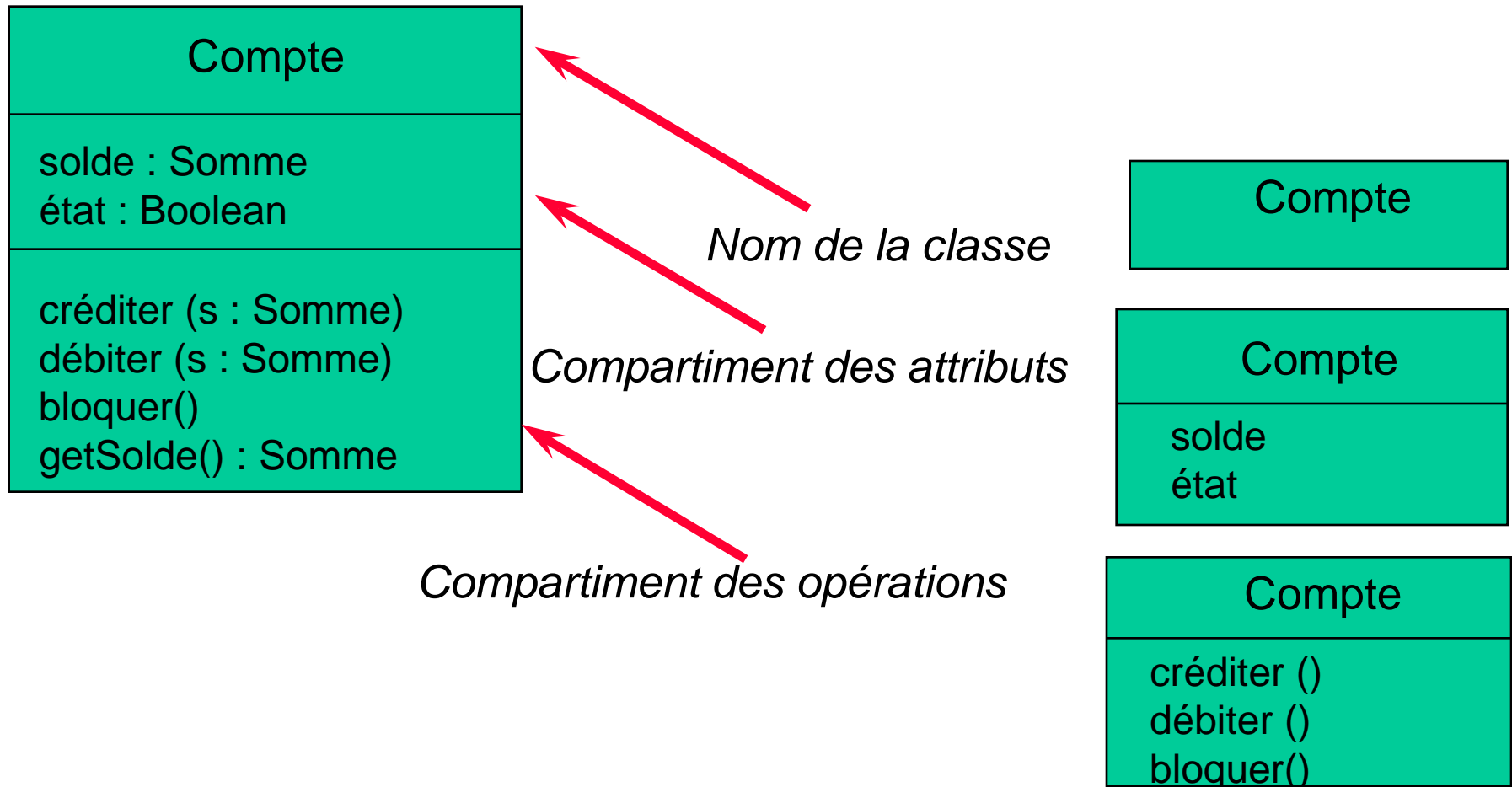
Point de vue  
Camarade



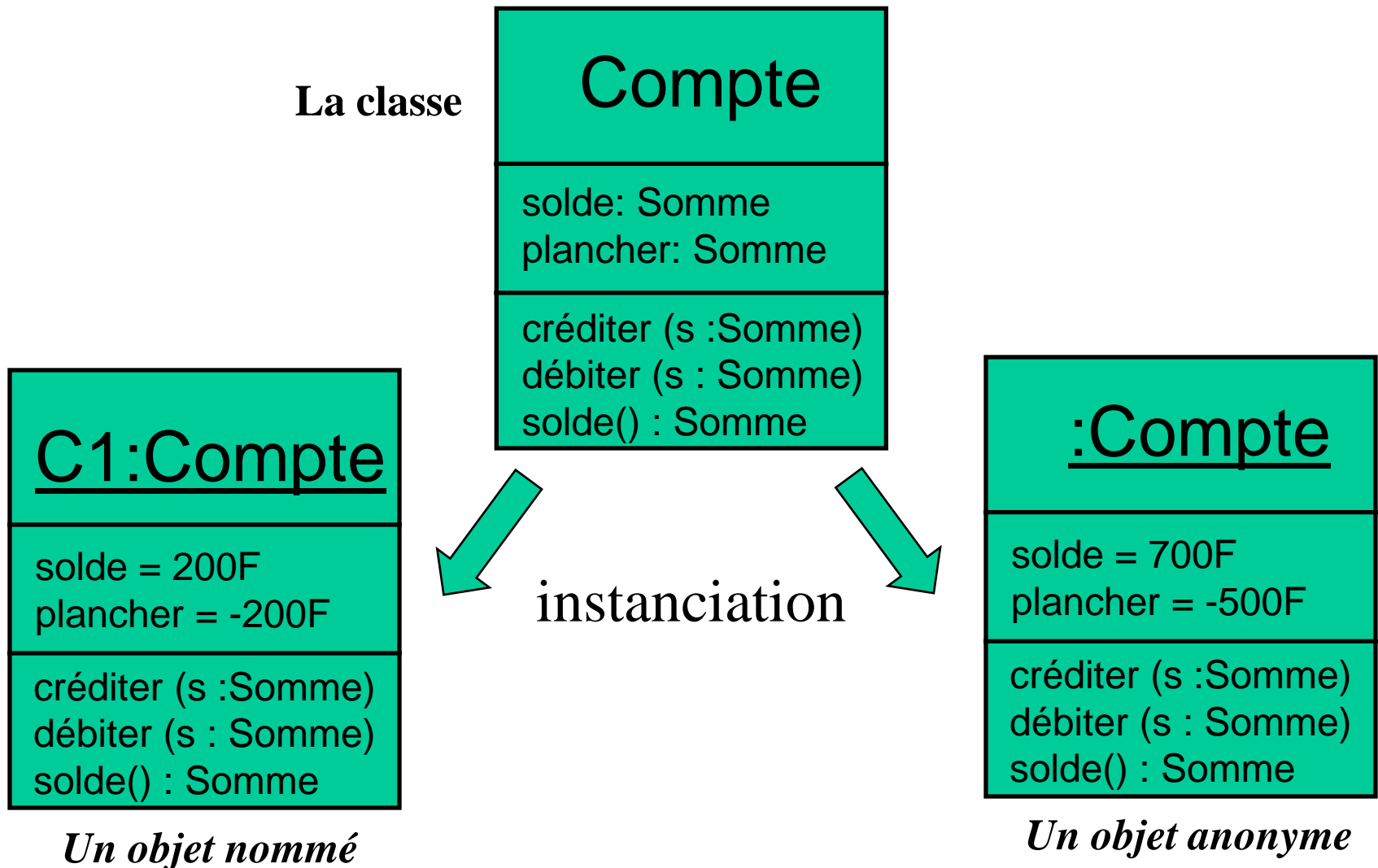
Point de vue  
Professeur



# 2.1. Représentation d'une classe en UML



# Le processus d'instanciation



# Modèle de données et transaction logique

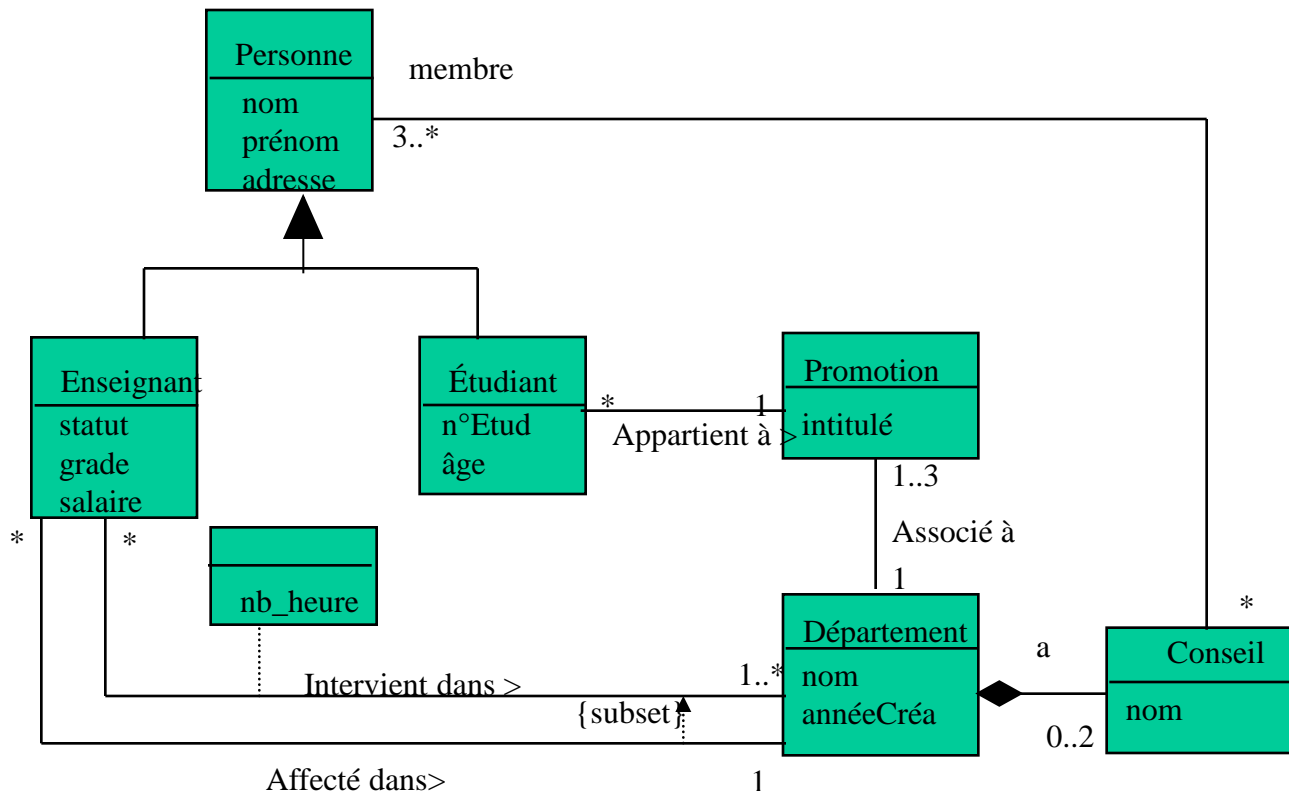
- Un MCD représente les données structurées dans leur état stable i.e. qu'il modélise l'état de l'ensemble des données à un moment t lorsqu'il y a aucune action ou manipulation en cours.
- Transaction logique (TL) :
- Les manipulations de données (ajout, mise à jour et suppression sur un ou plusieurs objets de classes différentes) sont exécutées par une suite d'actions élémentaires présumant la validation ultérieure des contraintes de multiplicité du MCD. Chaque TL est présumée en exécution concurrente avec d'autres transaction logiques. Les propriétés ACID sont assumées.

## Exemples:

- Ajout d'un employé : une seule action d'insertion dans la BD compose la transaction logique; à la fin de cette action les contraintes de multiplicité du MCD seront vérifiées.
- Ajout d'un employé et ajout d'un département dans la même transaction logique; à la fin de la TL les contraintes de multiplicité seront présumées vérifiées.
- Important: Si la cohérence est mise en péril, la TL doit être annulée.

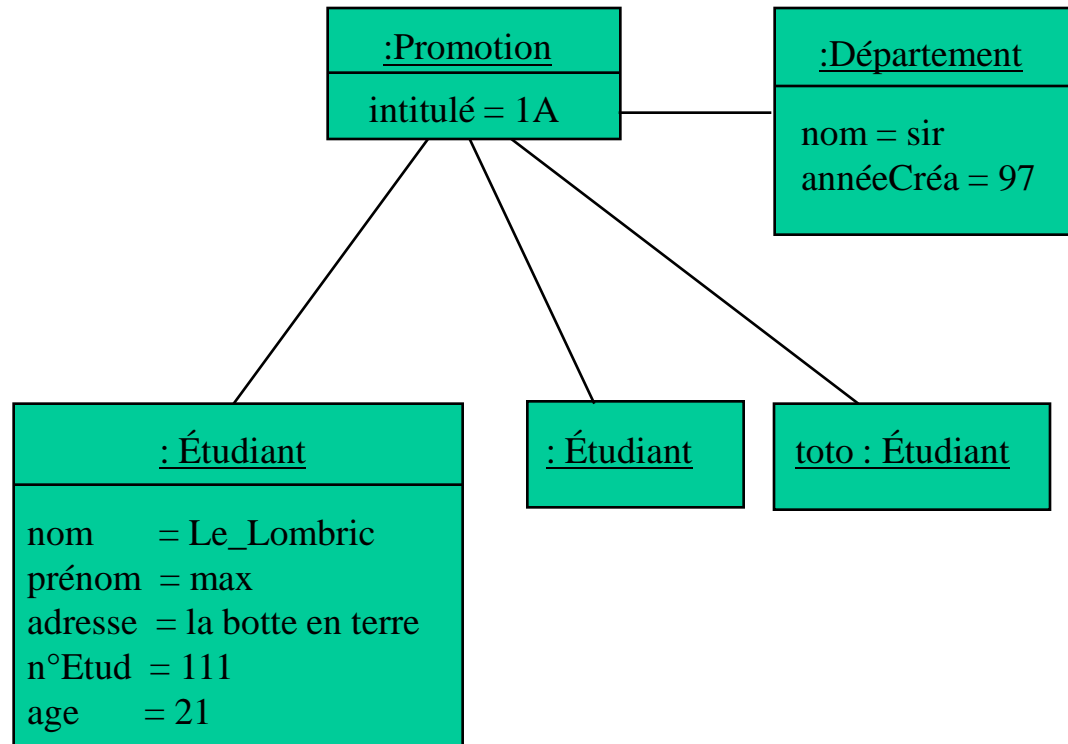
# Le Diagramme de Classes

- Structure statique du système en termes de classe et de relation entre ces classes au niveau conceptuel, d'analyse ou de conception



# Le Diagramme d'Objets

- Faciliter la compréhension des structures de données complexes décrites dans le DC
- Identifier les classes et les relations



# Association entre Classes

Diagramme de classes

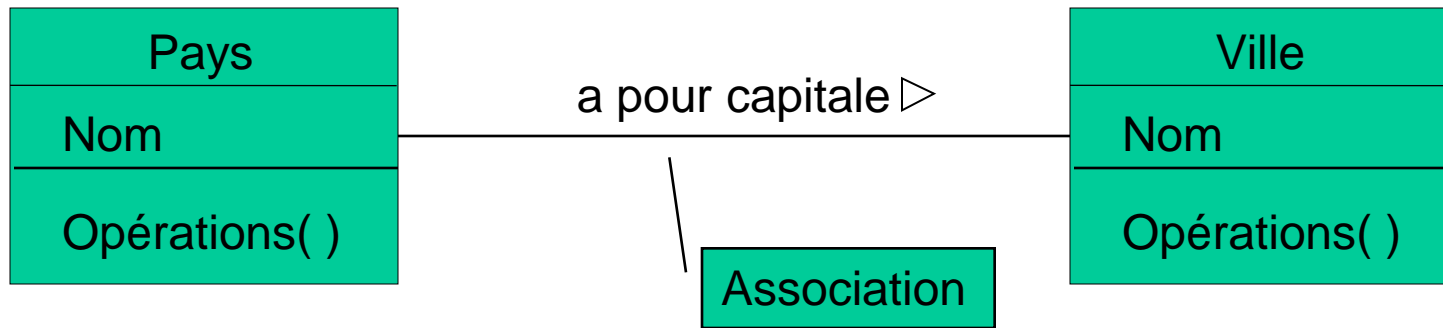
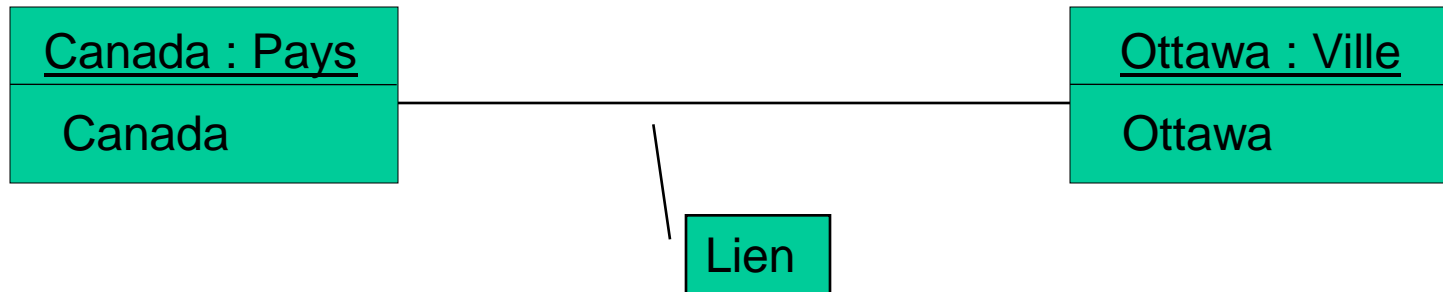


Diagramme d'objets = instance du diagramme de classes

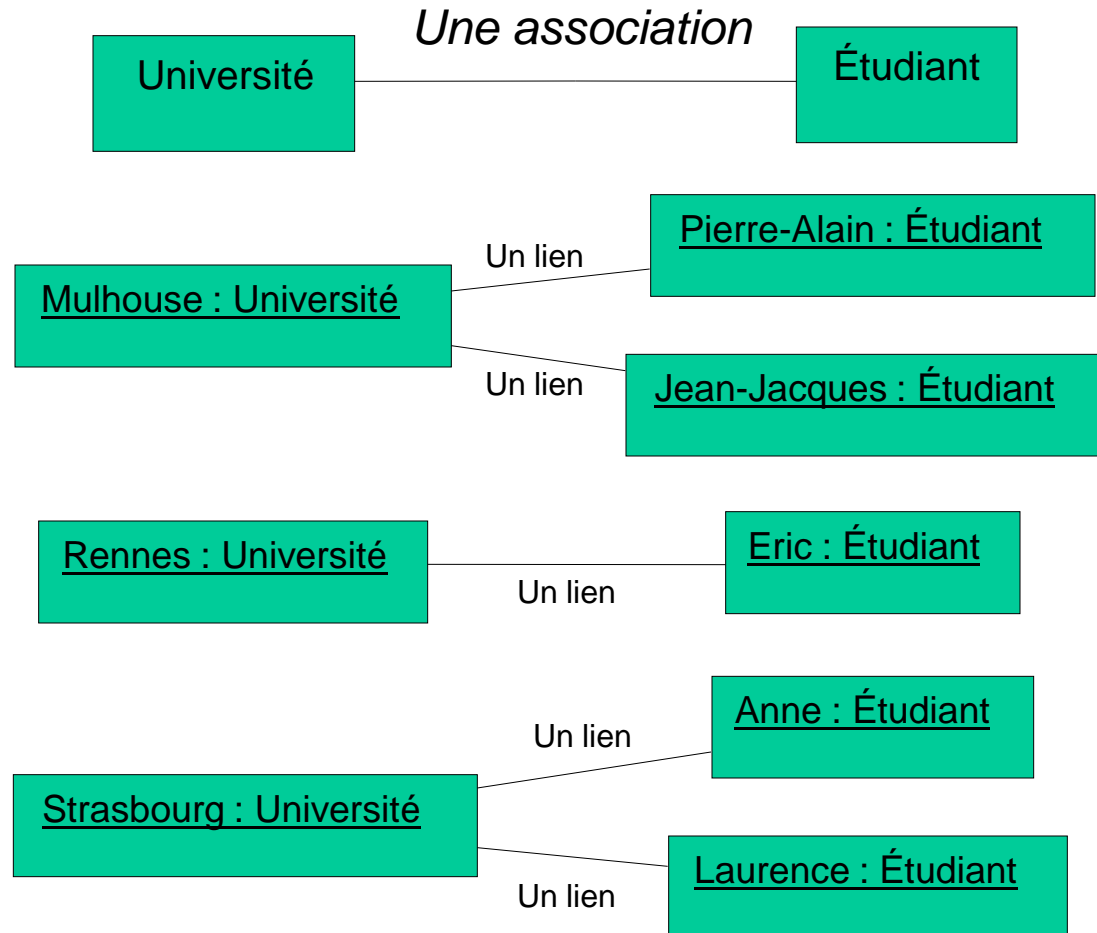




# Association entre Classes

- L'association exprime une connexion sémantique bidirectionnelle entre classes
- Une association est une abstraction des liens qui existent entre les objets instances des classes associées
- Les associations se représentent de la même manière que les liens
- Les associations montrent les dépendances entre les classes

# Association entre Classes



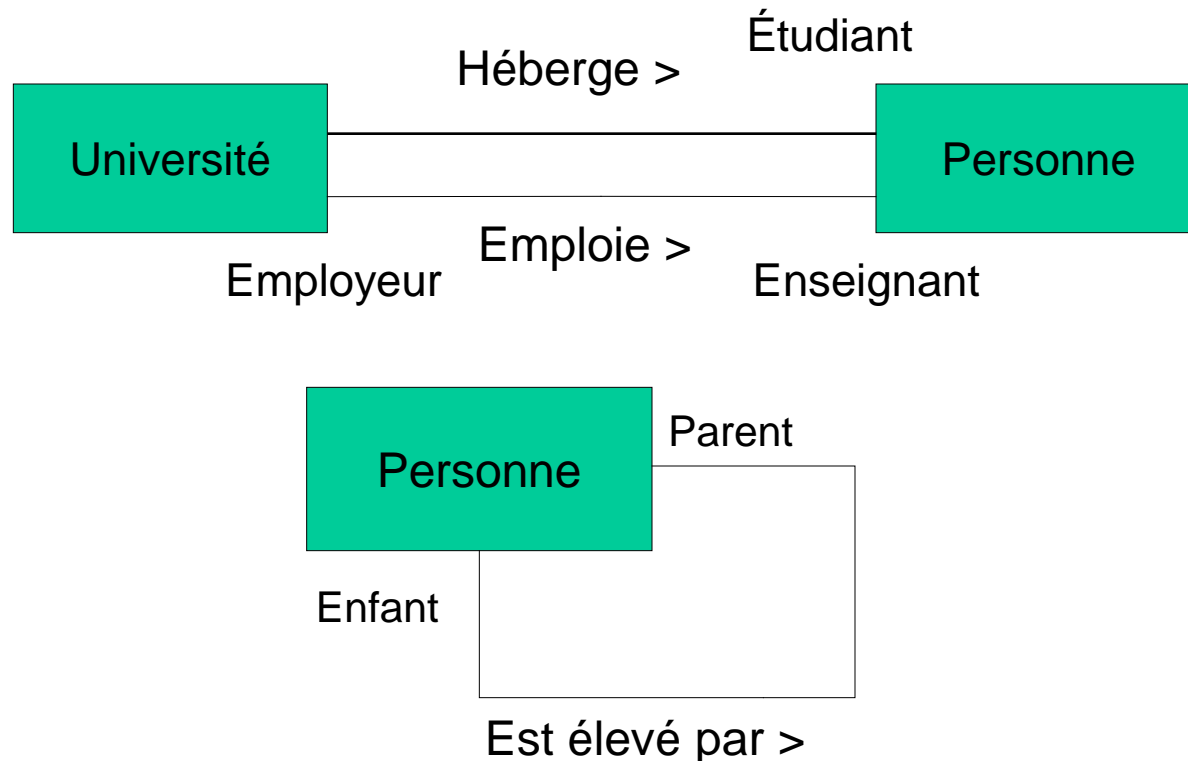
# Association entre Classes

Nommage des associations : indication du sens de lecture pour améliorer la lisibilité

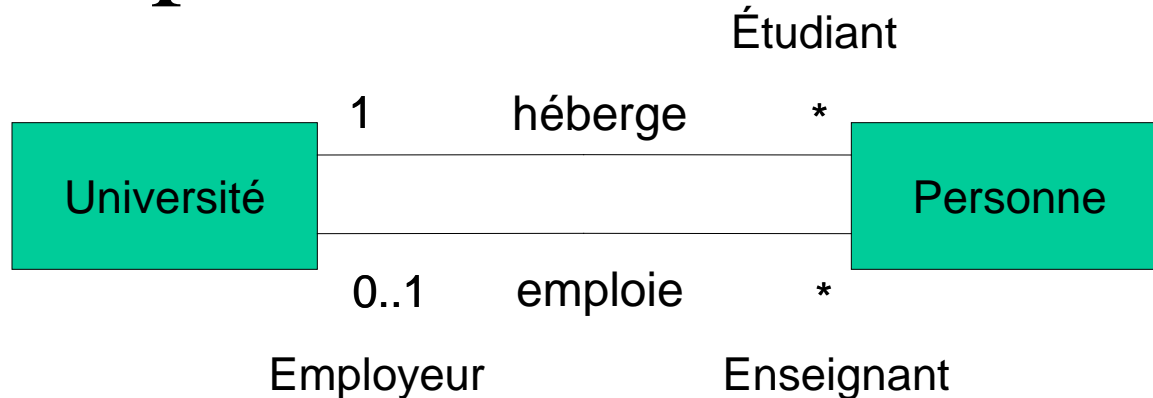


# Association entre Classes

Nommage des rôles : le rôle décrit une extrémité d'une association dans le cas de plusieurs associations



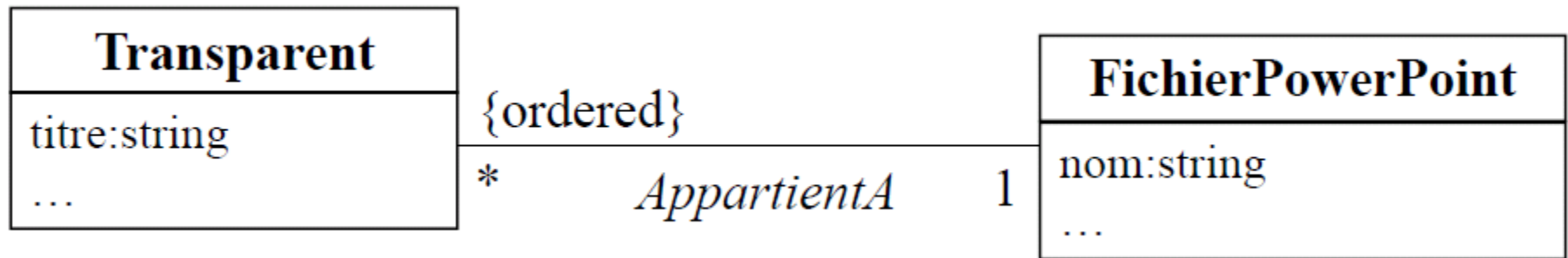
# Multiplicité d'une Association



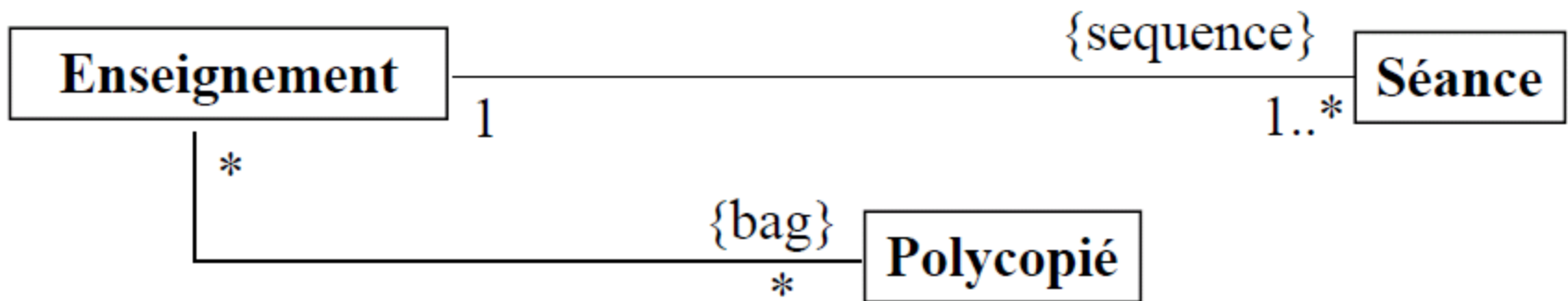
1	Un et un seul (eq 1..1)
0..1	Zéro ou un
M .. N	De M à N (entiers naturels)
*	Plusieurs (eq 0..*)
1 .. *	D'un à plusieurs

# Ordonnancement, *bags* et séquences à l'extrémité d'une association « plusieurs »

- Ordonnancement des objets situés à l'extrémité d'une association « plusieurs »



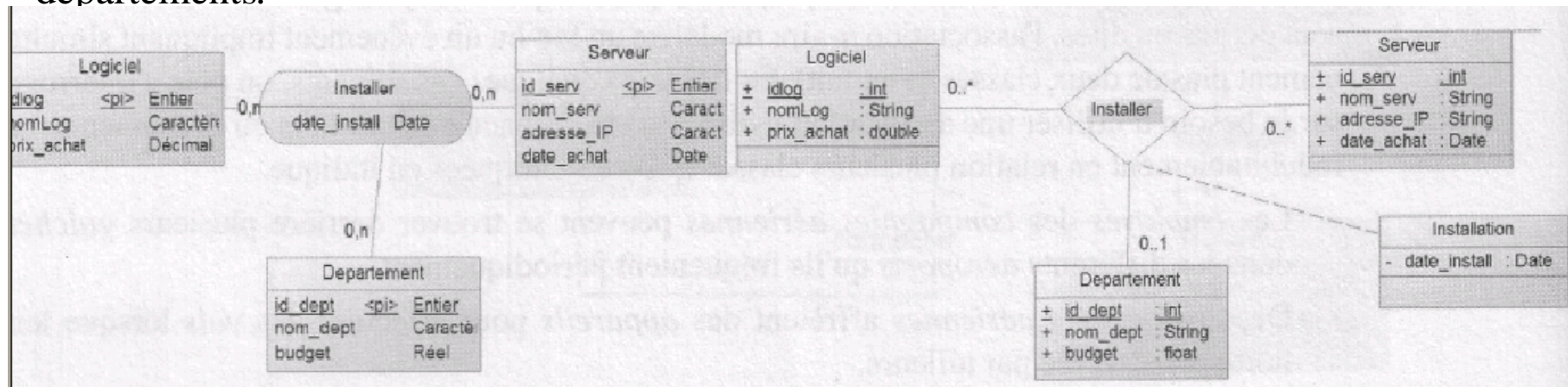
- Bag (sac) : collection non ordonnée avec autorisation de doublons



- Séquence : collection ordonnée avec autorisation de doublons

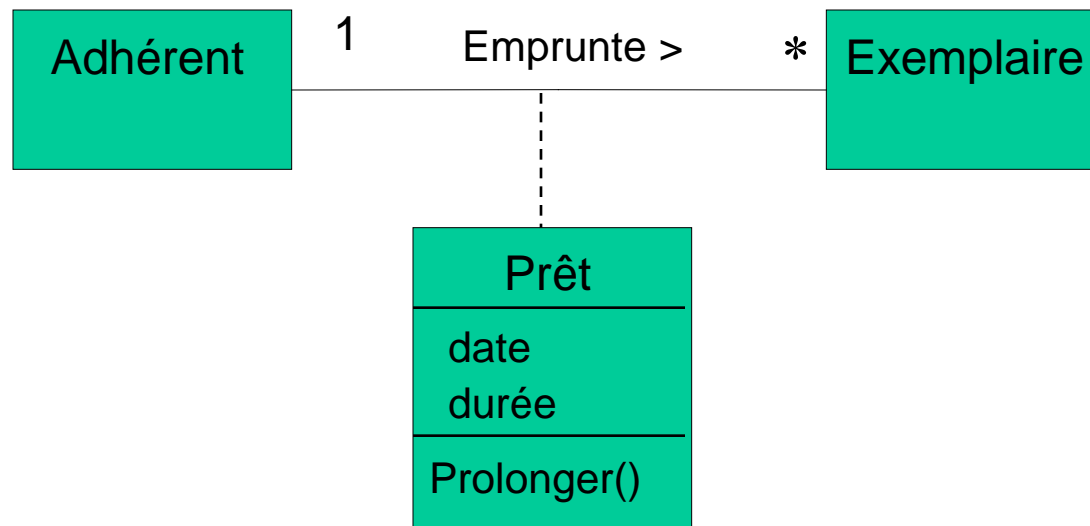
# Interprétation associations n-aires MERISE versus UML

- Avec le formalisme de Merise, les cardinalités sont lues du sens entité concernée --> entités connectées. Dans les autres formalismes entité-association et avec UML, les multiplicités d'une classe sont lues à partir des autres classes de l'association (sens classes connectées --> classe concernée).
- Aucune de ces deux lectures n'est l'inverse ou l'opposée de l'autre ; le sens est tout bonnement différent. Avec le formalisme Merise, vous ne pouvez pas déduire grand-chose des associations n-aires. La majorité d'entre elles ne contiennent que des cardinalités (0,n). La cardinalité (0,1) impose une décomposition en degré inférieur (une association 3-aire se décompose en deux associations binaires). Avec le formalisme UML, les associations n-aires sont davantage parlantes. La multiplicité (0..1) n'impose pas forcément une décomposition en degré inférieur.
- Dans l'exemple suivant, l'utilisation de l'association n-aire Installer permet d'exprimer que des logiciels sont installés un certain jour (date\_install) sur des serveurs par des départements.



# Les Classes-Associations

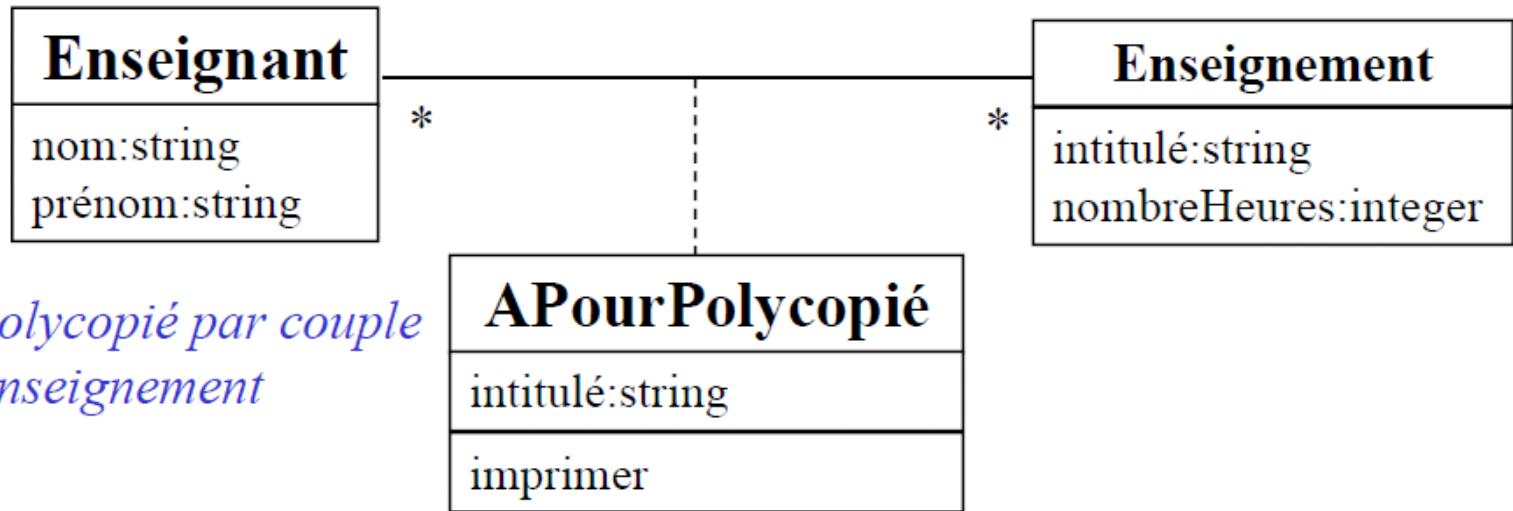
- Ajouter des opérations et des attributs à une association → créer une classe d'association
- Une instance d'association crée une instance de la classe d'association
- Exemple



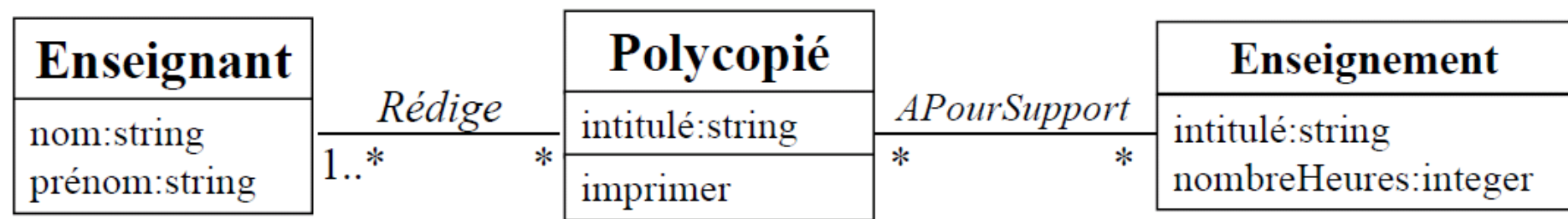


# Les Classes-Associations

- Ne pas confondre classe-association et association promue au rang de classe



*Il y a un seul polycopié par couple  
Enseignant / Enseignement*



*Un nombre quelconque d'occurrences de Polycopié pour chaque Enseignant et  
chaque Enseignement*

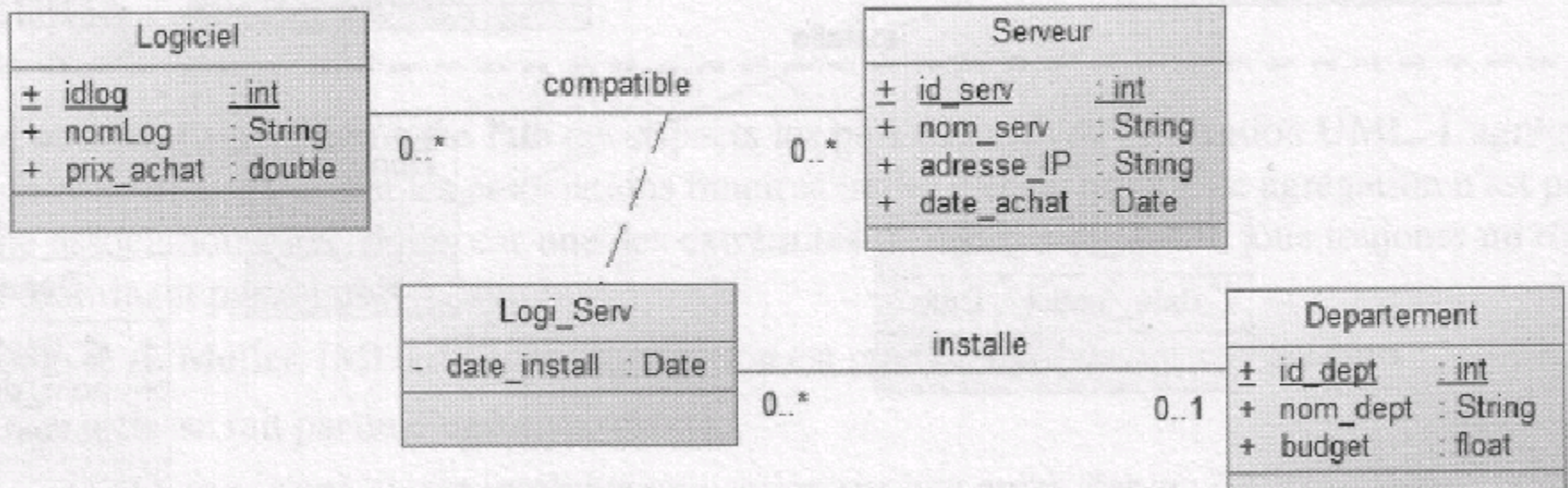


# Décomposition des associations n-aires en classes-associations

- Evitez d'utiliser des associations n-aires, car votre modélisation produira un schéma relationnel non contraint (décidez plutôt de ne pas contraindre un schéma au niveau logique). La majorité des associations n-aires doivent plutôt s'écrire, d'une manière hiérarchique, à l'aide d'une ou de plusieurs classes-associations.
- Lorsqu'il ne vous est pas possible de réduire le degré d'une association n-aire du fait qu'il n'existe aucun lien entre les différentes classes connectées, à part l'association elle-même, trouvez s'il existe (et c'est souvent le cas) une hiérarchie à travers une ou plusieurs classes-associations. En effet, une association n-aire exprime implicitement dans la majorité des cas une ou plusieurs contraintes d'inclusion.
  1. Vous devez choisir le couple de classes le plus « fort ». Dans certains cas, le choix s'impose facilement, dans d'autres, plusieurs couples peuvent être candidats.
  2. Reliez ces classes par une association *plusieurs-à-plusieurs*, si ce n'est pas possible, votre couple n'est pas solide (enfin, ce n'est pas ce que je veux dire !), piochez ailleurs (là non plus...).
  3. Attachez ensuite à cette association une classe-association qui contiendra éventuellement des attributs.
  4. La classe restante est à associer à la classe-association et toutes les possibilités de multiplicités sont permises.

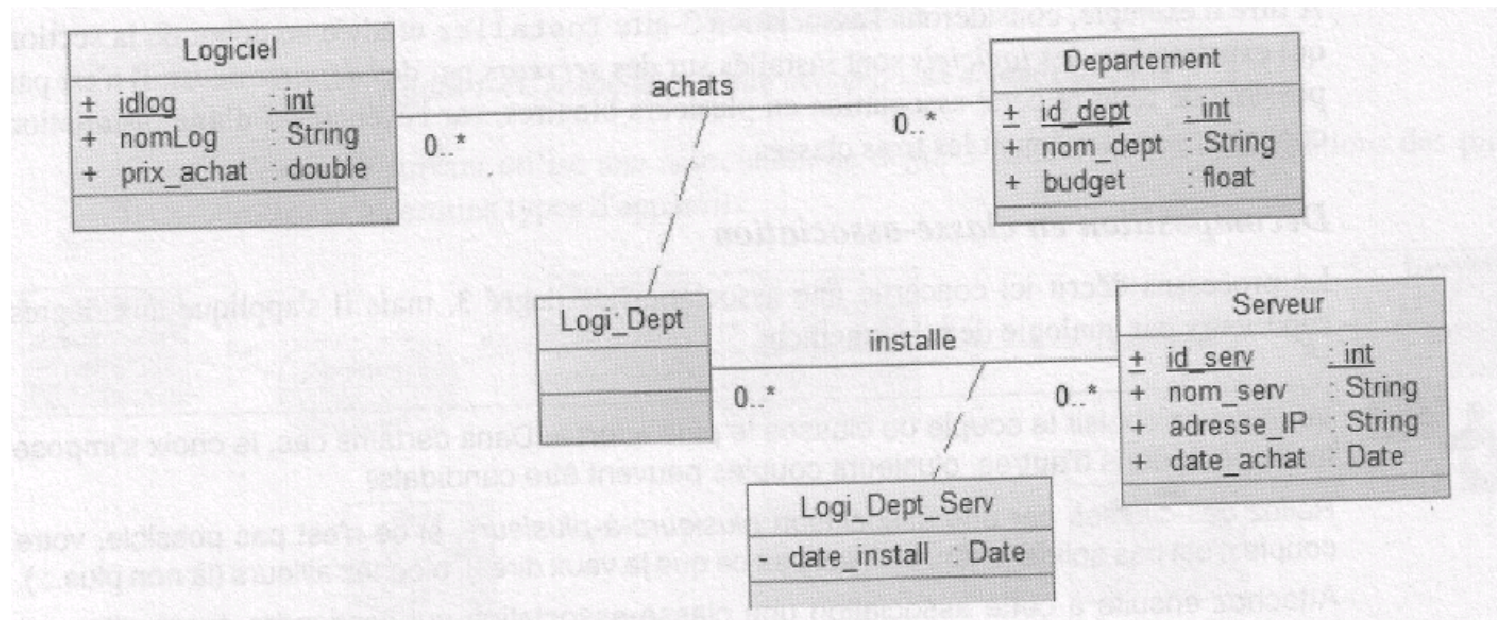
# Décomposition des associations n-aires en classes-associations : l'exemple Installer, choix 1

- Un premier choix peut se porter sur les couples (*logiciel, serveur*) qui recensent toutes les compatibilités (et évitent ainsi les incompatibilités). La classe-association contient la date d'installation (qui dépend en fait du seul couple). L'association d'installation concerne bien simultanément les trois classes en reliant un couple (*logiciel, serveur*) au *département* acteur de l'événement.



# Décomposition des associations n-aires en classes-associations : l'exemple Installer, choix 2

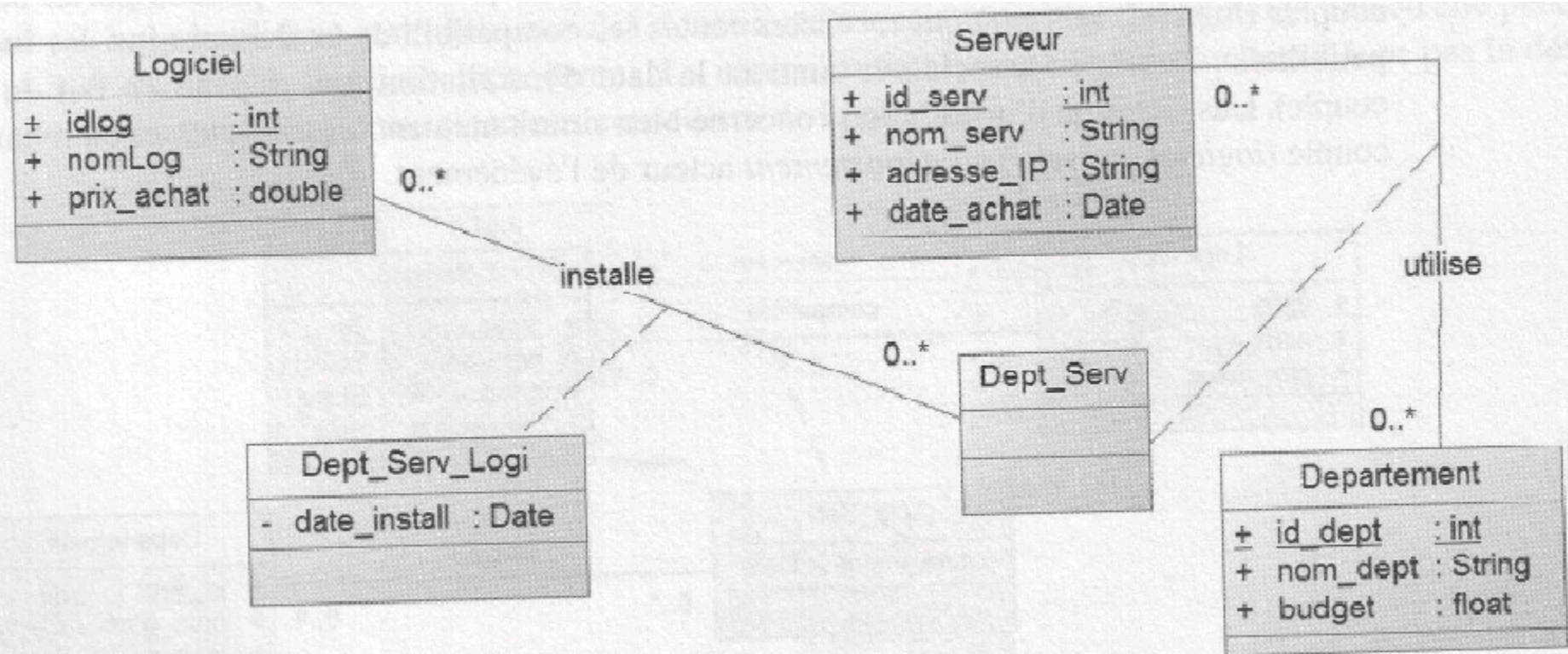
- Une deuxième branche de l'alternative consiste à considérer les couples (*logiciel*, *département*) qui recensent tous les achats (évitant ainsi d'installer un logiciel non acquis par le département). La classe-association contient la date d'installation (qui dépend en fait du seul couple). L'association d'installation concerne toujours simultanément les trois classes en reliant un couple (*logiciel*, *département*) aux *serveurs* dont le but est l'hébergement. La date d'installation doit se trouver au niveau de cette association avec cette modélisation.





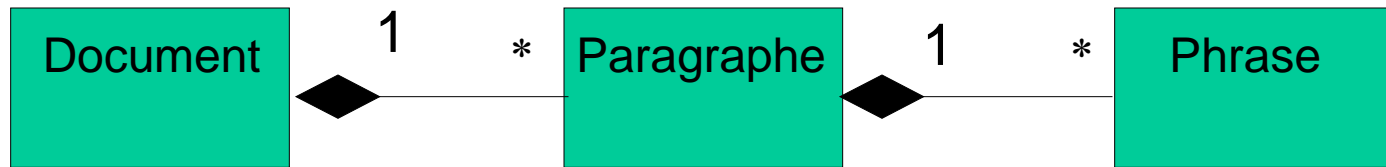
# Décomposition des associations n-aires en classes-associations : l'exemple Installer, choix 3

- Une troisième branche de l'alternative consiste à considérer les couples (*département*, *serveur*) qui recensent toutes les autorisations (évitant ainsi d'installer un logiciel sur un serveur non accessible par le département). Cette solution est symétrique à la précédente.



# Agrégation / Composition

Propriété de l'agrégation : transitivité (si  $A \Rightarrow B \Rightarrow C$  alors  $A \Rightarrow C$ ) mais non symétrique (si  $A \Rightarrow B$ ,  $B \Rightarrow A$ )



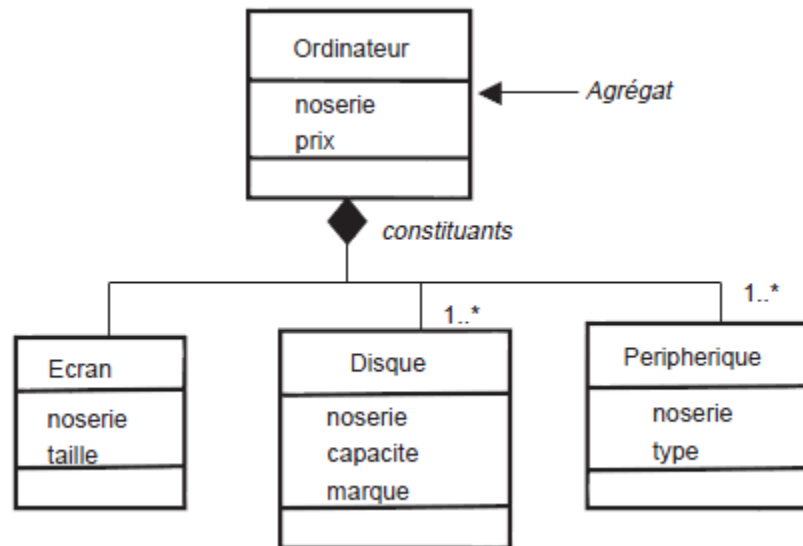
Si multiplicité de l'agrégat = 1, destruction de l'agrégat  $\Rightarrow$  destruction des composants

# Agrégations d'UML

- Les agrégations d'UML représentent des associations qui ne sont pas symétriques et pour lesquelles l'une des extrémités joue un rôle prédominant par rapport à l'autre. L'agrégation concerne seulement les associations binaires (réflexives ou non).
- Il est préférable d'utiliser une agrégation lorsqu'une classe fait partie d'une autre classe ou lorsqu'une action sur une classe implique une action sur une autre classe.
- UML définit deux formes d'agrégation :
  - La *composition (composite aggregation)* requiert qu'un objet appartienne au plus à une composition d'objets à un instant donné. Cette appartenance peut changer au cours du temps. La composition implique en outre une forme de propagation entre le composite et le composant (en particulier la destruction du composite entraînera obligatoirement la destruction de ses composants). Cette agrégation est représentée par un *losange noirci du côté du rôle de la classe composite* appelée « agrégat ». L'agrégation de composition ne s'impose que lorsque l'association est de type *composite/composant ou fait partie de*.
  - L'*agrégation partagée* autorise qu'un objet appartienne simultanément à différentes compositions d'objets. Cette agrégation est représentée par un *losange clair du côté du rôle de la classe concernée* par l'association appelée « agrégat ». Les agrégations partagées sont un peu plus difficiles à définir, elles renforcent le couplage d'une association binaire et interviennent lorsque la composition ne s'applique pas et que des objets sont fortement dépendants par rapport à d'autres objets dans le cadre de l'association.

# Composition UML

- Modélisons un ordinateur qui est composé d'un écran, d'un ou de plusieurs disques et de différents périphériques en utilisant une agrégation de composition. Les composants (objets des classes Ecran, Disque et Peripherique) et les composites (objets de la classe Ordinateur) sont ici considérés comme des pièces réelles. La destruction d'un ordinateur entraînera la destruction de ses composants.

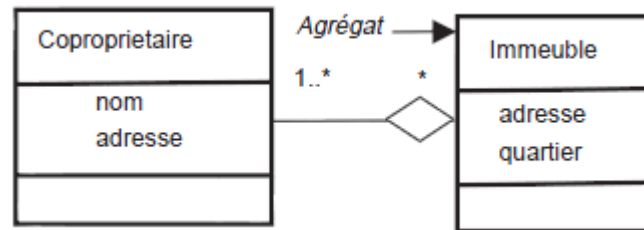


- Au niveau de la base de données, la composition se programme avec SQL à l'aide de la propagation de l'action de suppression via les clés étrangères (ON DELETE CASCADE) ou par déclencheurs.
- Les associations qualifiées doivent être mises en œuvre par le concept d'agrégation forte (composition)



# Agrégation partagée UML

- Le losange clair indiquerait que les composants seraient considérés comme des types de pièces et non comme des pièces réelles. Chaque objet des classes Ecran, Disque et Peripherique pourrait ainsi être partagé entre différents objets de type Ordinateur. La destruction d'un ordinateur n'entraînerait pas nécessairement la destruction de ses composants.
- Considérons aussi l'existence d'un copropriétaire n'a de sens que si l'immeuble existe. Il ne s'agit pas ici d'une relation composite/composant. La suppression d'un objet de type immeuble n'entraînera pas forcément la destruction de ses copropriétaires (pour ceux qui sont copropriétaires dans d'autres immeubles).

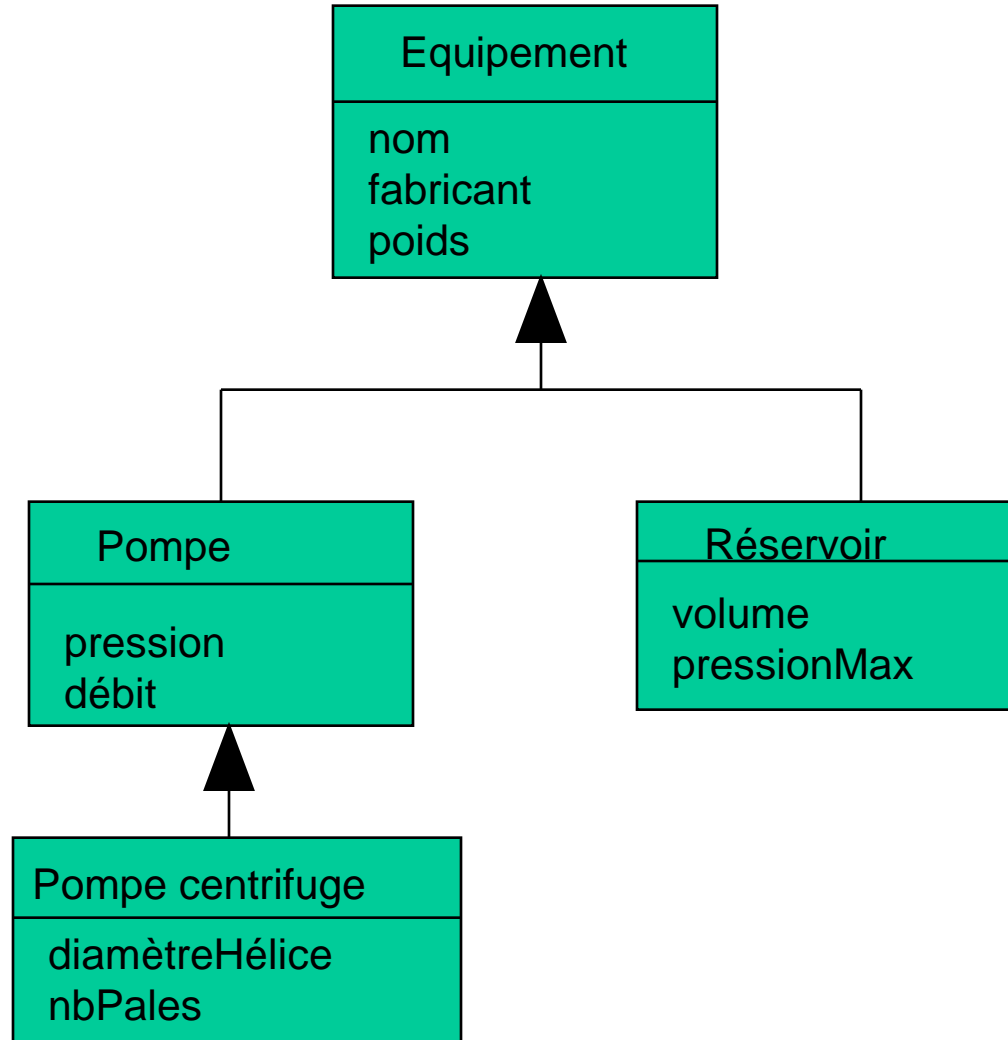


- Notion assez subjective, l'agrégation relève plus de la conception détaillée que de la modélisation. Elle se traduira au niveau physique par programmation de déclencheurs ou de contraintes SQL (de vérification CHECK ou de répercussion de clé étrangère CASCADE).

# Relation de Généralisation

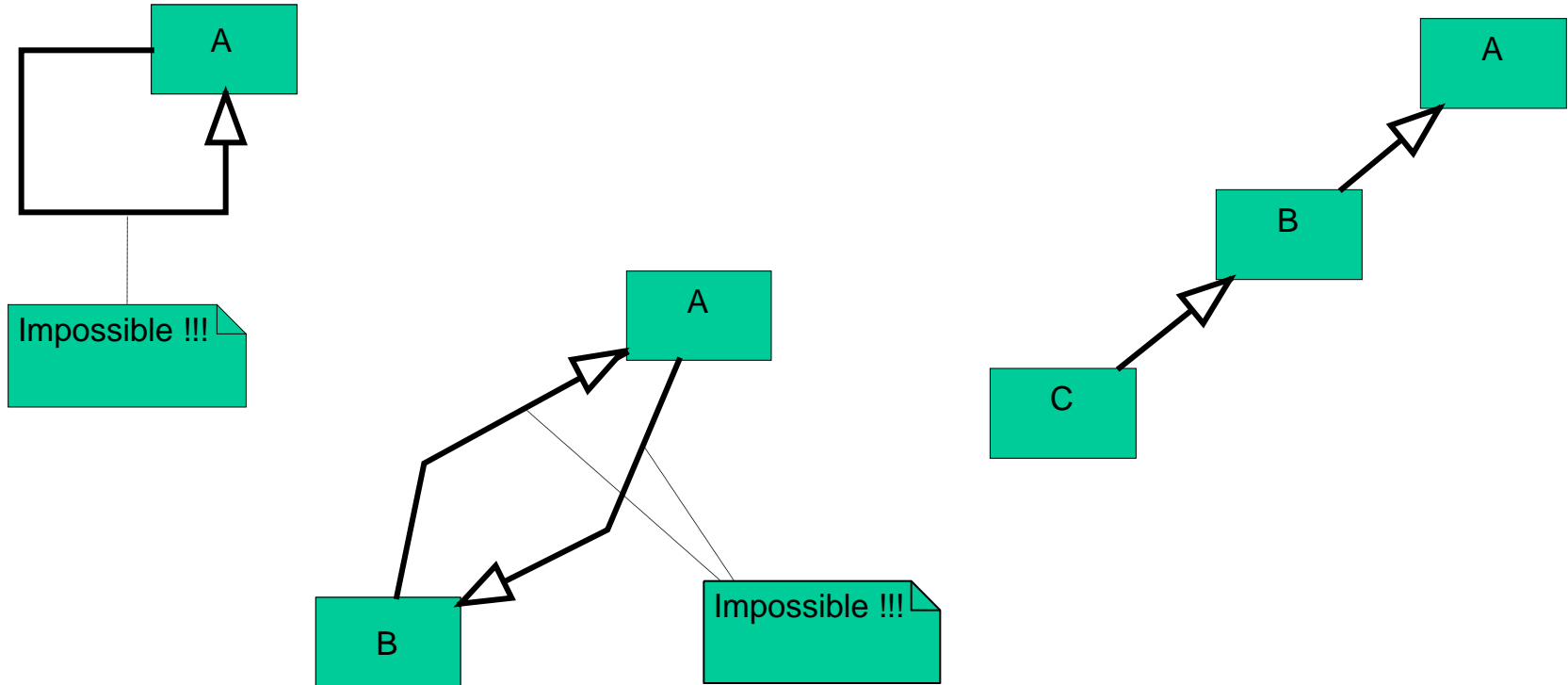
- La relation de Généralisation est un concept fondamental de la modélisation objet.
- Cette relation entre classes est un mécanisme permettant de définir une classe (on parle de sous-classe ou de classe fille) à partir d'une autre classe (on parle de super-classe) tout en l'affinant.
- L'affinement peut consister en l'ajout d'attributs, l'ajout ou la redéfinition d'opérations.
- La sous-classe présente en plus de ses propres propriétés toutes les propriétés de sa super-classe. Un objet instance de la sous-classe possèdera donc toutes les propriétés (attributs, opérations, relations, contraintes) définies dans la super-classe plus toutes les propriétés définies dans la sous-classe.
- La relation de généralisation est souvent traduite sémantiquement par « est un ».

# Représentation graphique de la généralisation



# Propriétés de la relation de Généralisation

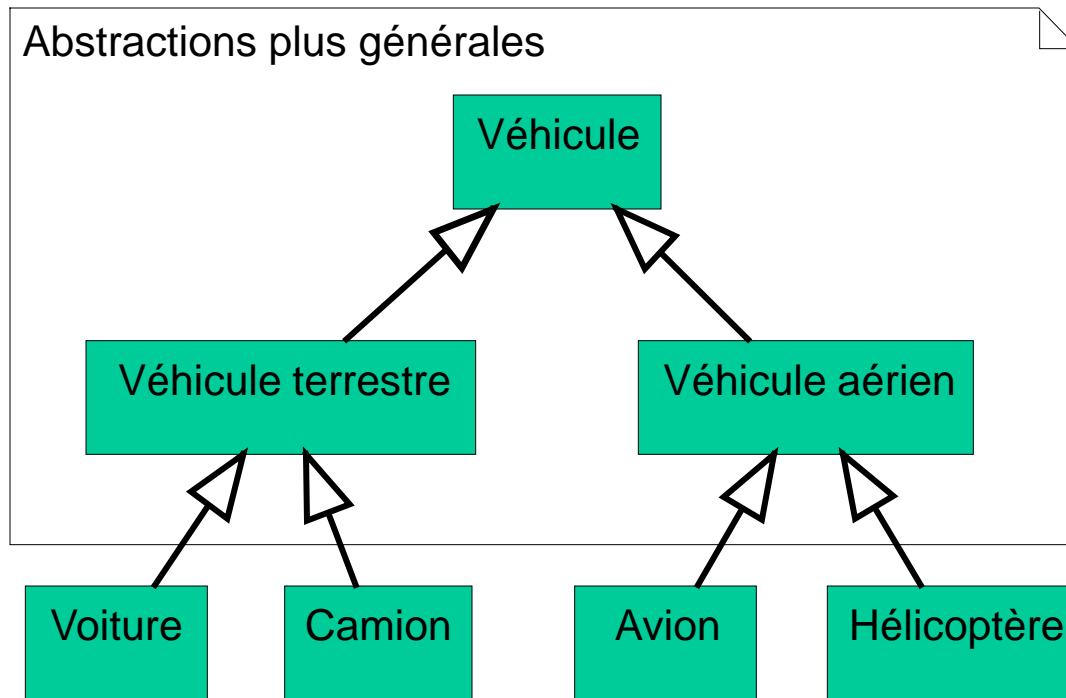
- Non réflexive, non symétrique, transitive



# Généralisation

Généralisation = factoriser les éléments communs, c'est-à-dire les attributs et les méthodes

La généralisation est opérée après identification des classes



# Généralisation et Spécialisation

- Il y a deux façons d'appréhender la relation de Généralisation lors du développement d'un logiciel.
  - Partant d'une classe existante on définit une sous-classe capturant des particularités d'un ensemble d'objets non discriminés par la classe existante. La sous-classe est une **spécialisation** de la super-classes. L'acte de spécialisation est à la base de la réutilisation.
  - Partant d'un ensemble de classe existante on factorise à l'aide d'une super-classe les propriétés communes à ces classes dans une classe plus générale. La super-classe ainsi définie est une **généralisation** des classes existantes. La généralisation est un acte difficile mais de grand profit (architecture plus facilement maintenable).

# Généralisation simple et multiple

- Il y a deux modes d'utilisation de la relation de généralisation.
- **Généralisation simple** : une classe ne peut avoir au plus qu'une seule super-classe.
- **Généralisation multiple** : une classe peut avoir un nombre quelconque de super-classes.
- UML propose les deux modes. Le choix de se restreindre au mode simple est souvent lié à la contrainte d'un langage cible ne supportant que l'héritage simple (comme JAVA).

# Généralisation et Agrégation

- Ne pas confondre « agrégation » et « généralisation »
  - L 'agrégation met en relation des objets → chaque objet est l 'instance d 'une classe
  - La généralisation met en relation des classes → un objet est simultanément instance de la super-classe et une instance de la sous-classe
  - Agrégation → « est composé de », relation « et »
  - Généralisation → « est une sorte de », relation « ou »



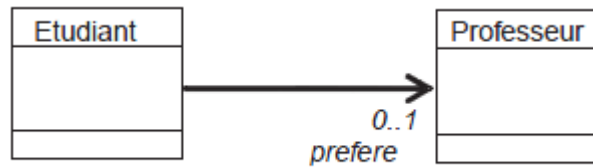
# Généralisation et Agrégation

## Exemple

- Une lampe de bureau est de type fluorescent ou à incandescence. Elle possède un socle, un abat-jour, un interrupteur et un câble. La lampe à incandescence est équipée d'une douille à barillet tandis que la lampe fluorescente est équipée d'un ballast, d'un starter et de deux douilles rotatives
- Diagramme des classes faisant apparaître l'héritage et l'agrégation ?

# Association navigable

- Par défaut, une association UML est navigable dans les deux sens. Cela signifie que si, dans le cadre de la relation, un objet *A est relié à un objet B*, par définition *B est aussi relié à A* (dans le cadre de cette même relation).
- UML 2 permet d'exprimer dans un diagramme que les instances d'une classe ne connaissent pas les instances d'une autre bien qu'étant reliées entre elles dans le cadre de l'association.



- La propriété de navigation est vérifiée dans la grande majorité des associations. La réduction de la portée d'une association sera réalisée en phase d'implémentation (par programmation).
- Il n'y a pas de sens à définir une association précisant la navigation des deux côtés simultanément.
- Pour une base de donnée objet-relationnelle, il y a un intérêt car les associations navigables ne sont pas significatives du point de vue des données, mais des traitements.



# Université de Bretagne Sud

*Informatique*



## Génie Logiciel avec UML



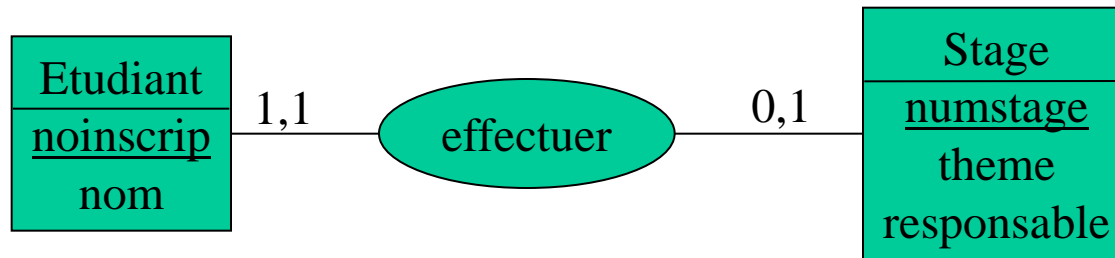
### Chapitre 5

### Equivalence DC/MCD

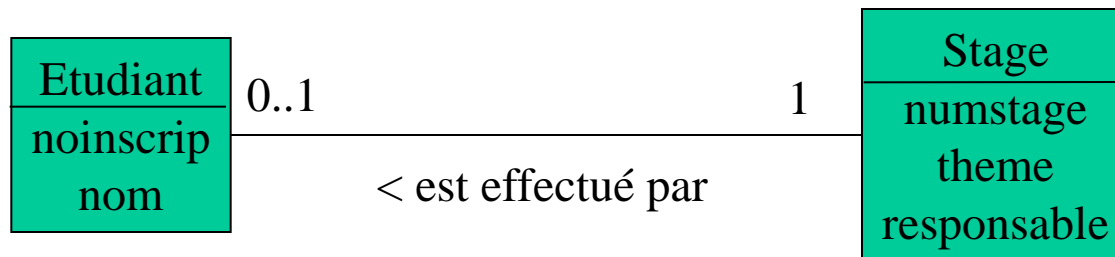
[Retour au plan](#)

# Associations 1-1

Merise

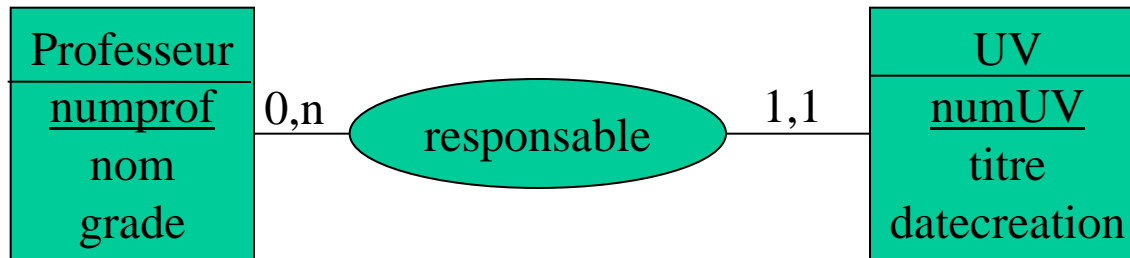


UML

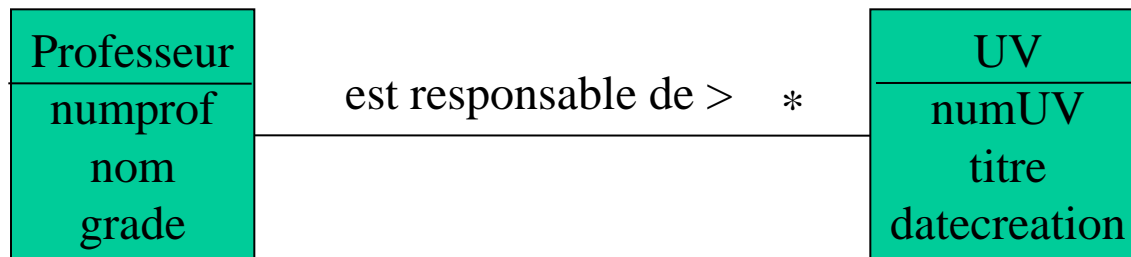


# Associations 1-N

Merise

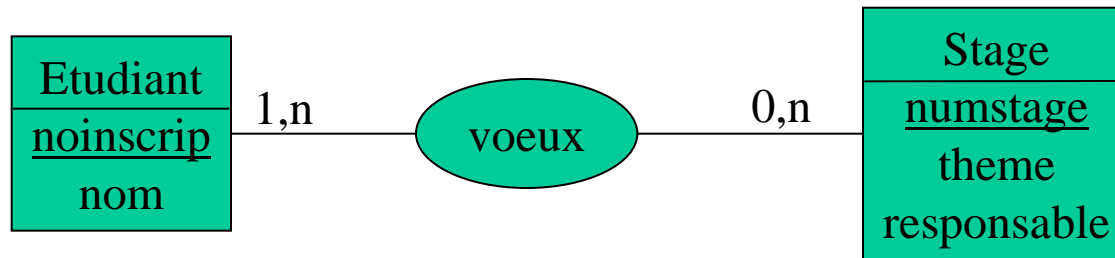


UML

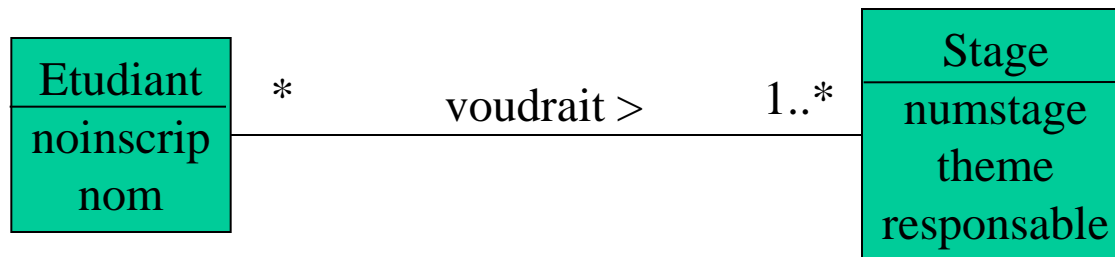


# Associations N-N sans propriété

Merise



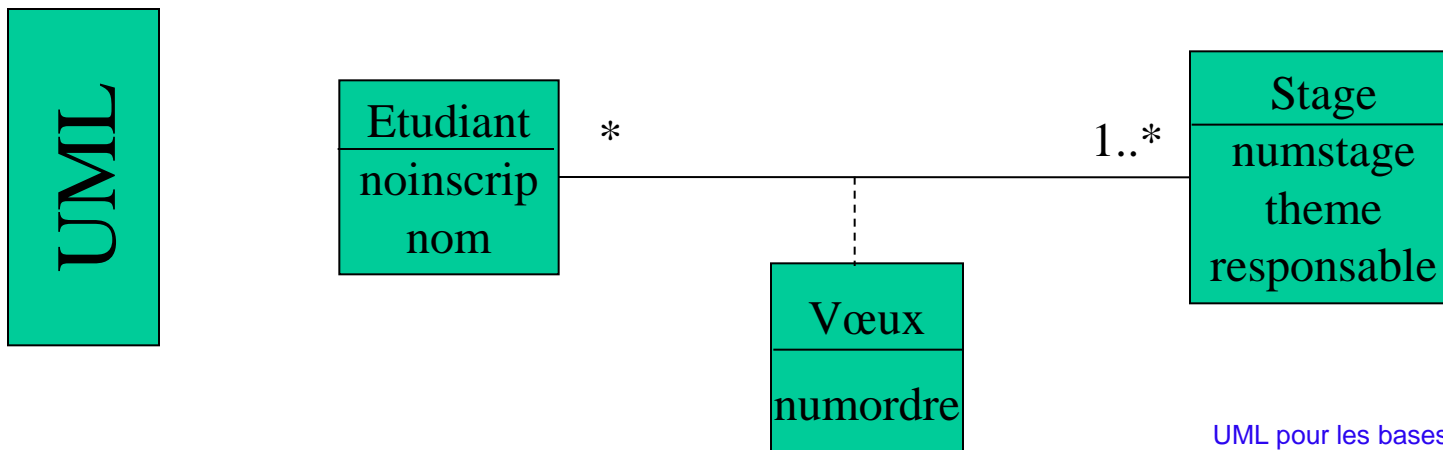
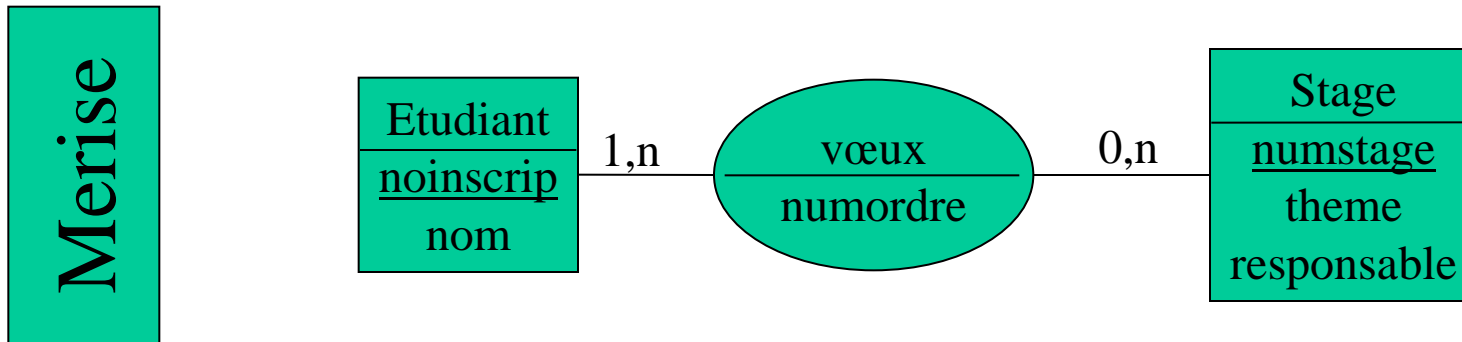
UML



# Associations N-N avec propriétés

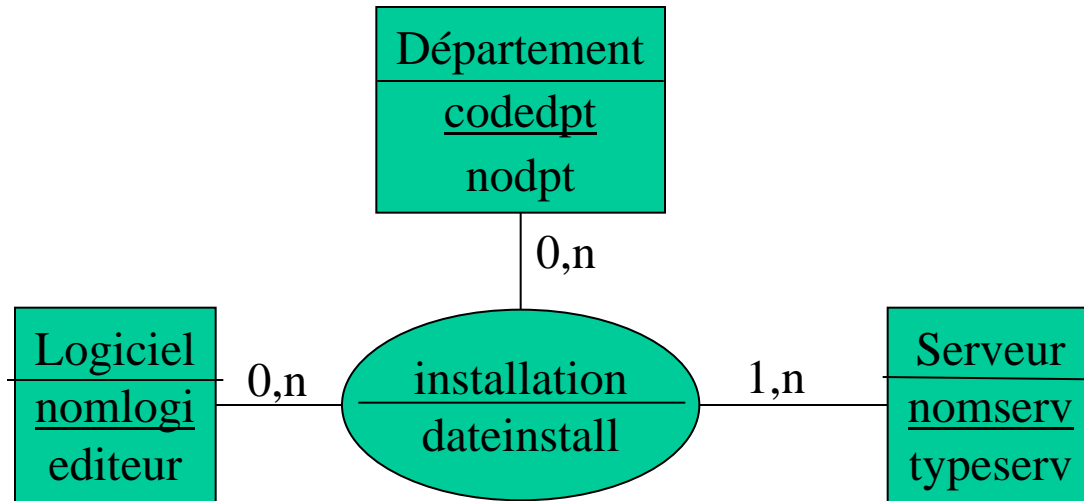
Une association *plusieurs-à-plusieurs avec attributs* est représentée sous UML par une *classe-association*.

Cette classe-association contient les attributs de l'association et est connectée au lien d'association par une ligne en pointillé.

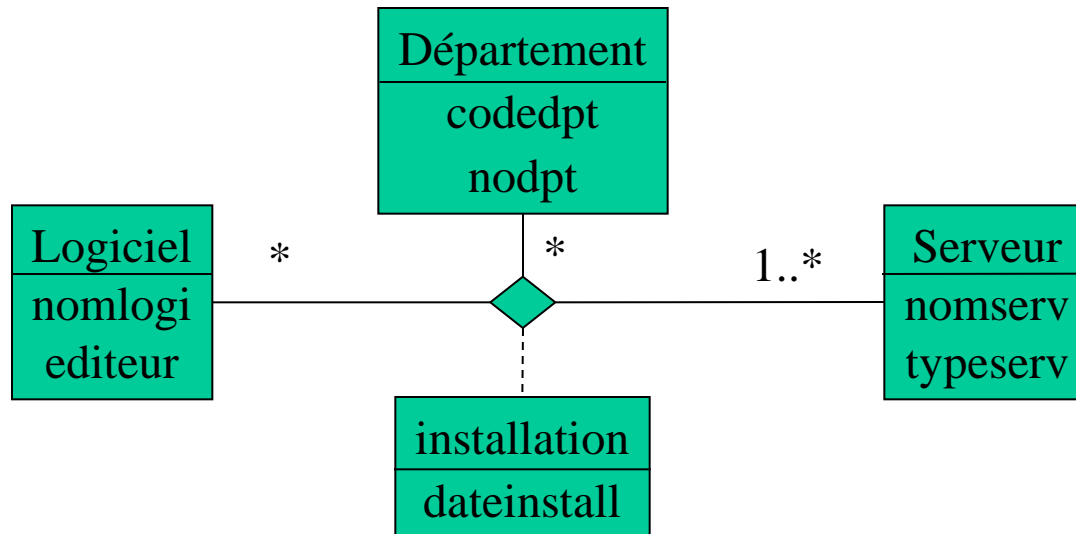


# Association n-aire

Merise



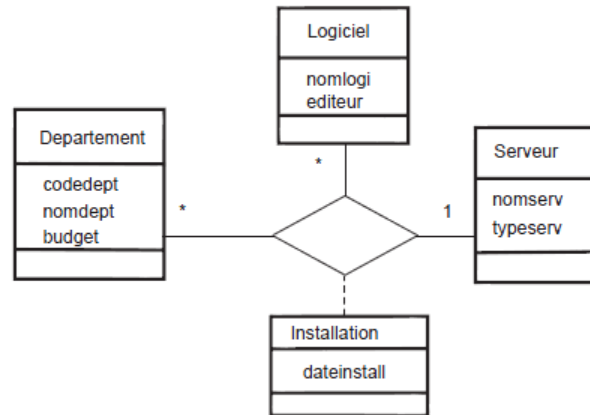
UML



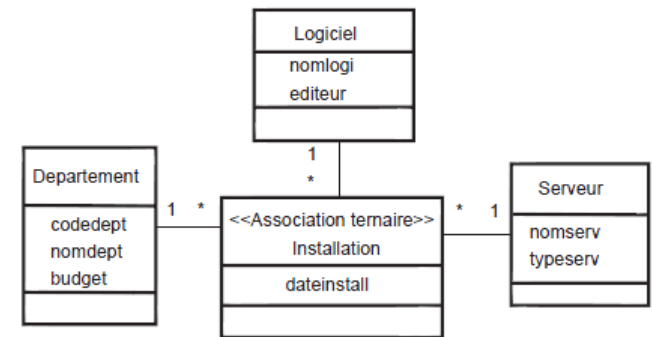


# Association n-aire

- Une association *n-aire sans contrainte* se représente avec UML soit par un losange (symbole de l'association) qui connecte les *n* classes et une classe-association, soit par une classe stéréotypée reliée aux *n* classes.
- Une association *n-aire avec contrainte* se représente plus facilement avec UML par une ou plusieurs classes-associations reliant les *n* classes.
- **Association en losange**

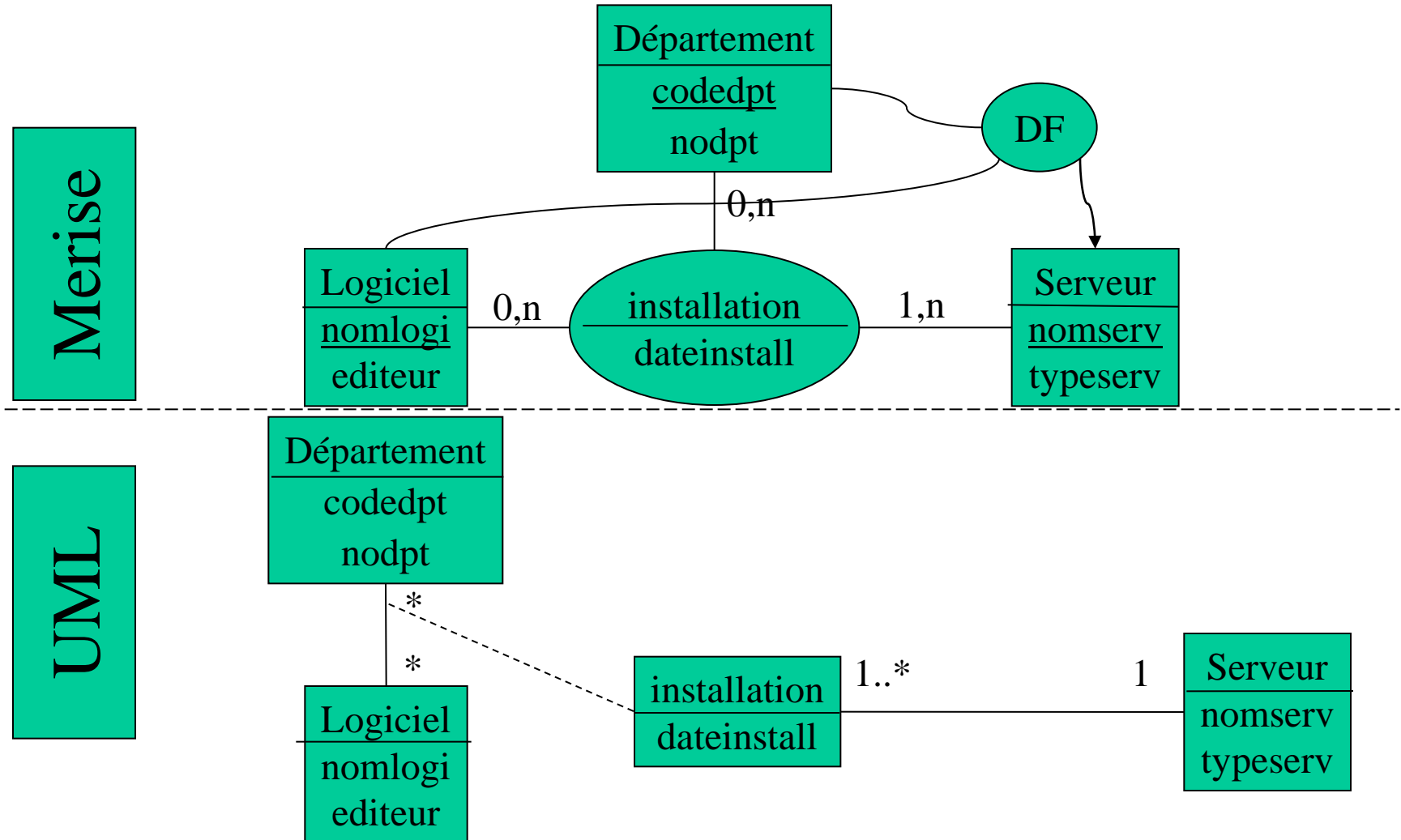


- **Classe stéréotypée**



Cette notation n'est pas conseillée à moins d'utiliser un outil qui ne prend pas en compte le symbole losange de l'association et de vouloir toutefois modéliser une association *n-aire sans* contrainte.

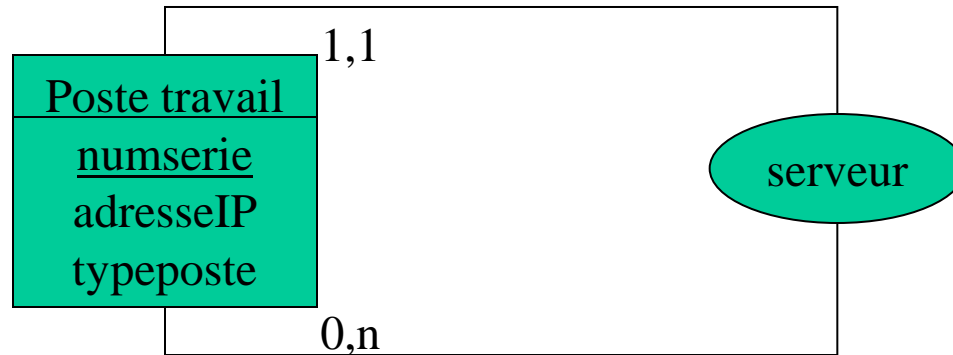
# Association n-aire avec DF



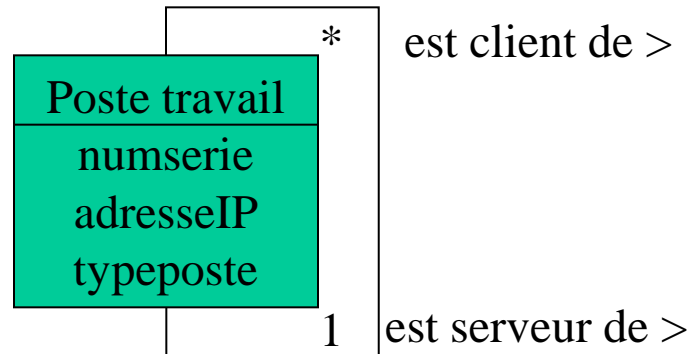
La classe-association permet de prendre en compte explicitement la contrainte d'unicité qui concerne l'installation des logiciels : *un logiciel d'un département n'est installé que sur un seul serveur.*

# Association réflexive 1-N

Merise

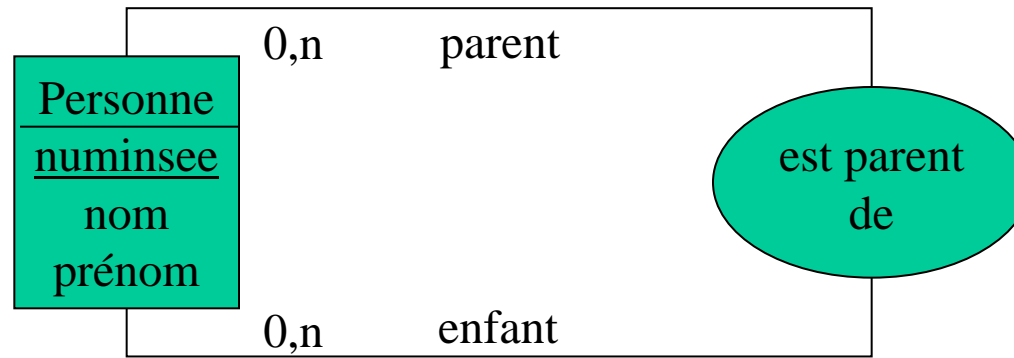


UML

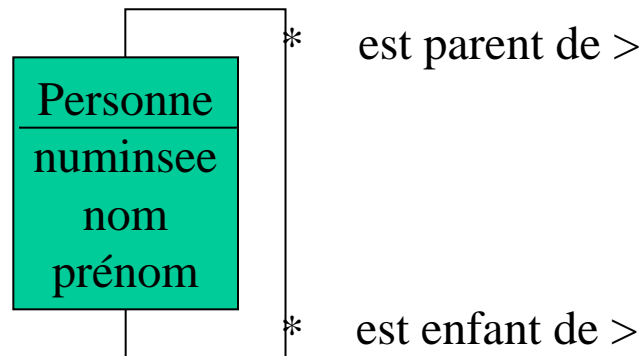


# Association réflexive N-N

Merise

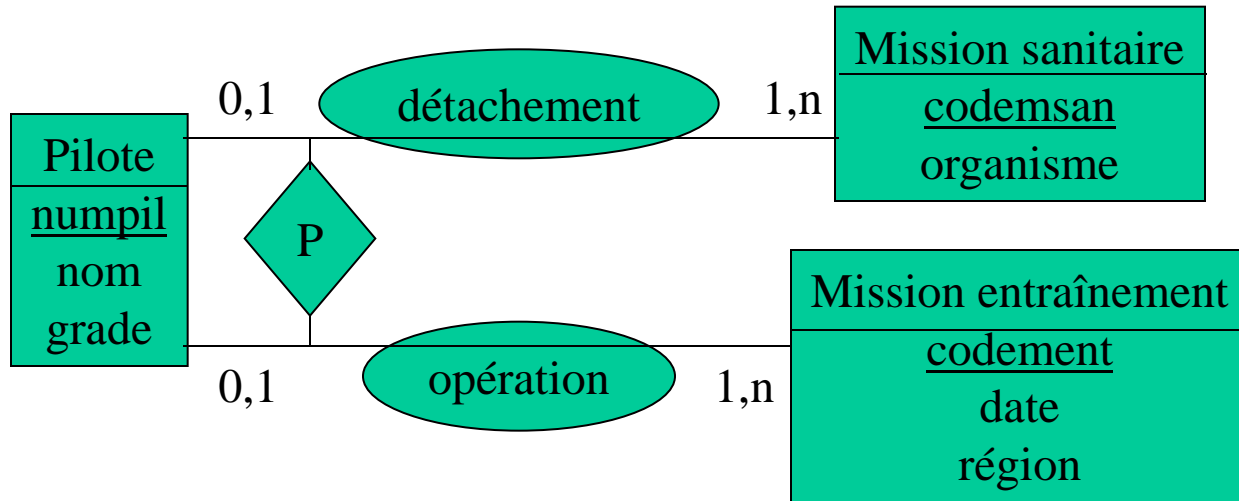


UML

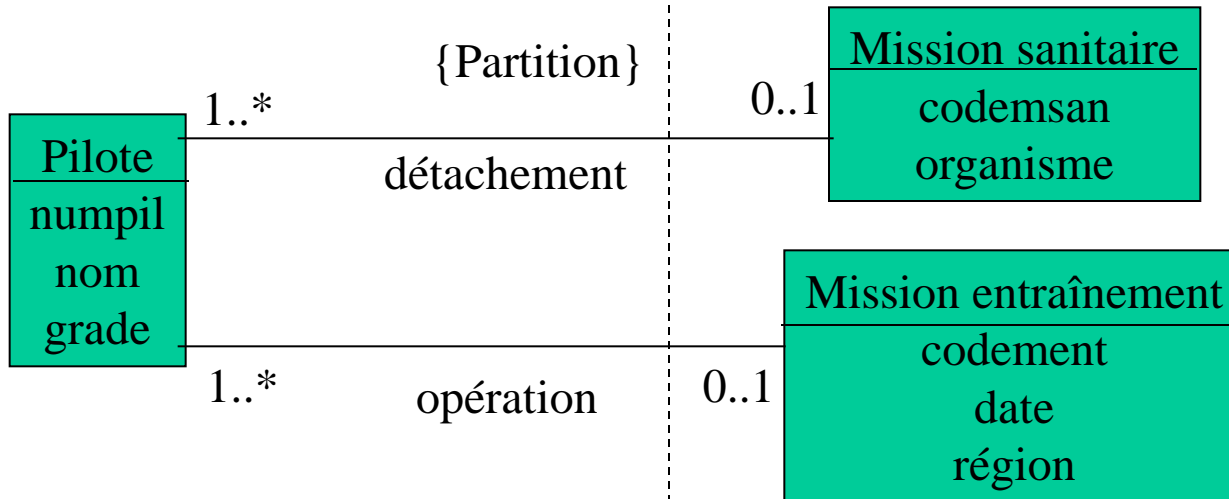


# Contrainte de partition

Merise



UML



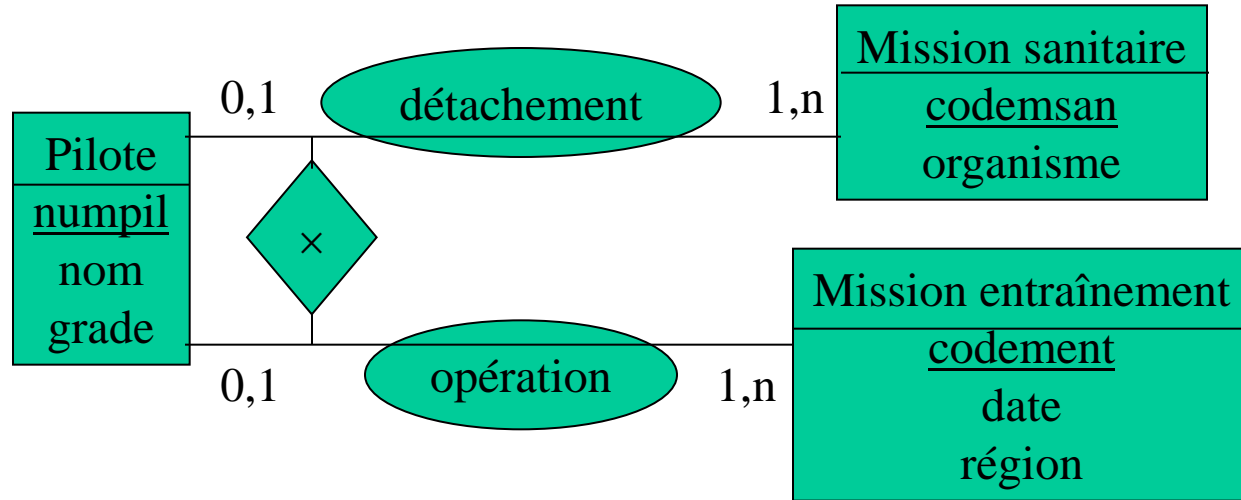
{xor} est possible.

En OCL: inv:

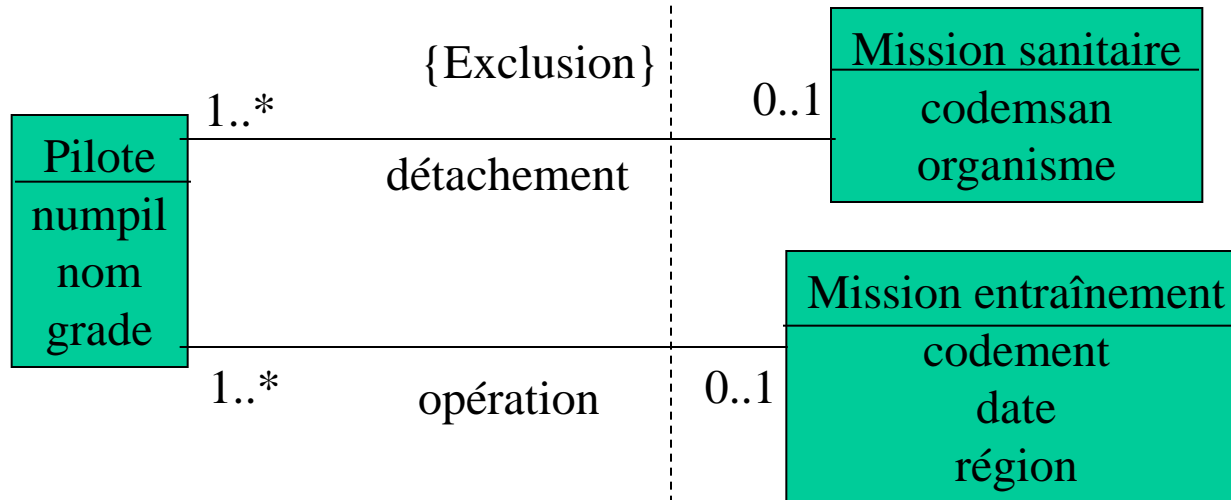
```
((detachement->isEmpty() and operation->notEmpty()) or
(detachement->notEmpty() and operation->isEmpty()))
```

# Contrainte d'exclusion

Merise



UML

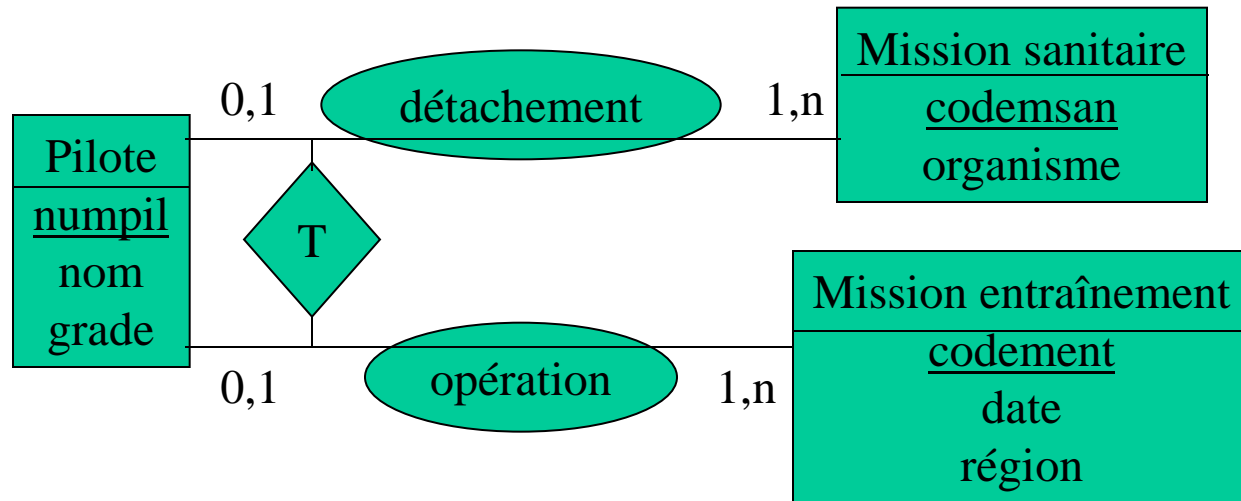


En OCL: inv:

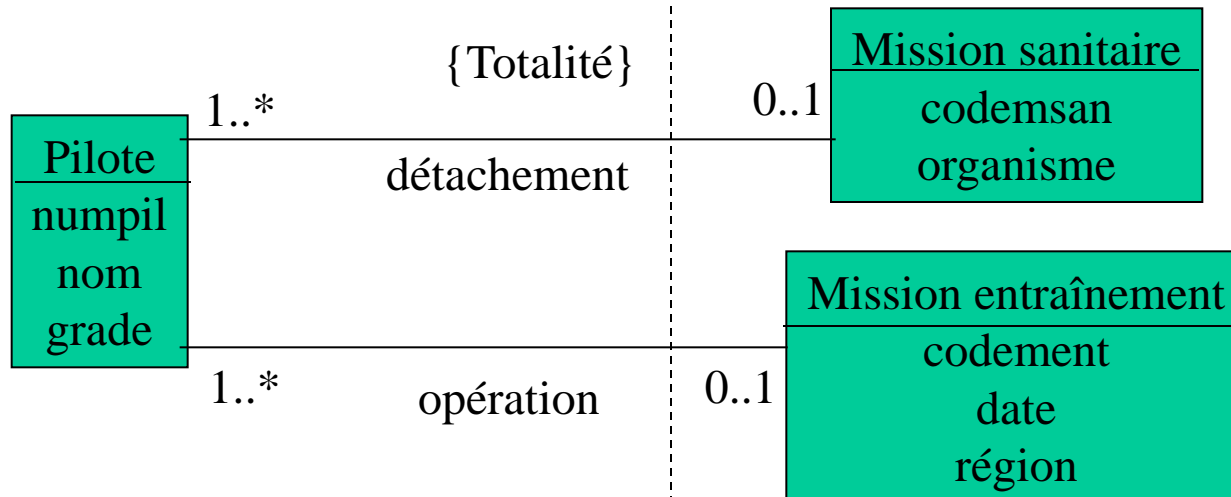
```
((detachement->isEmpty() and operation->notEmpty()) or
(detachement->notEmpty() and operation->isEmpty())) or
((detachement->isEmpty() and operation->isEmpty())
```

# Contrainte de totalité

Merise

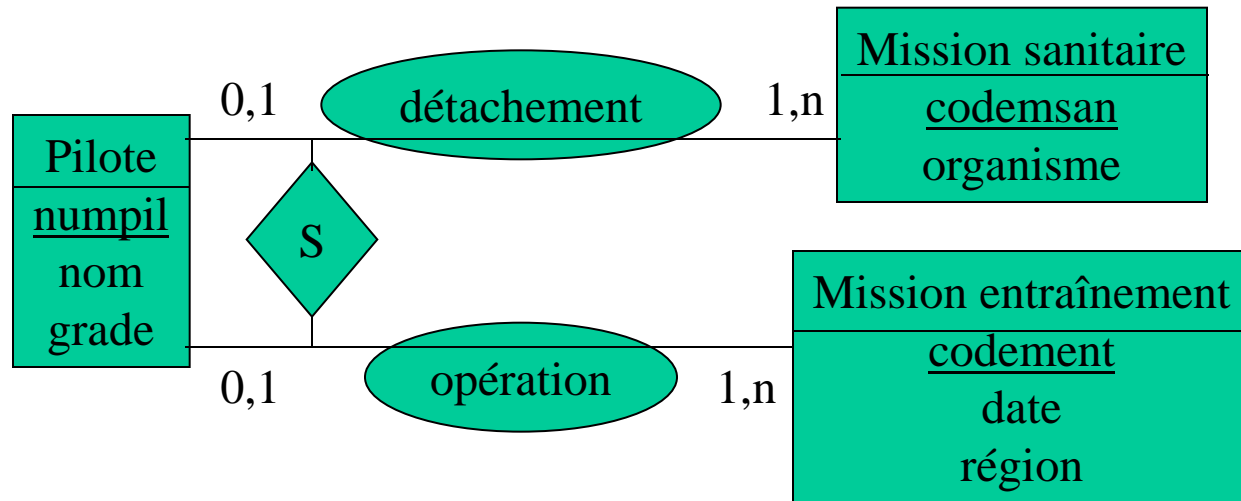


UML

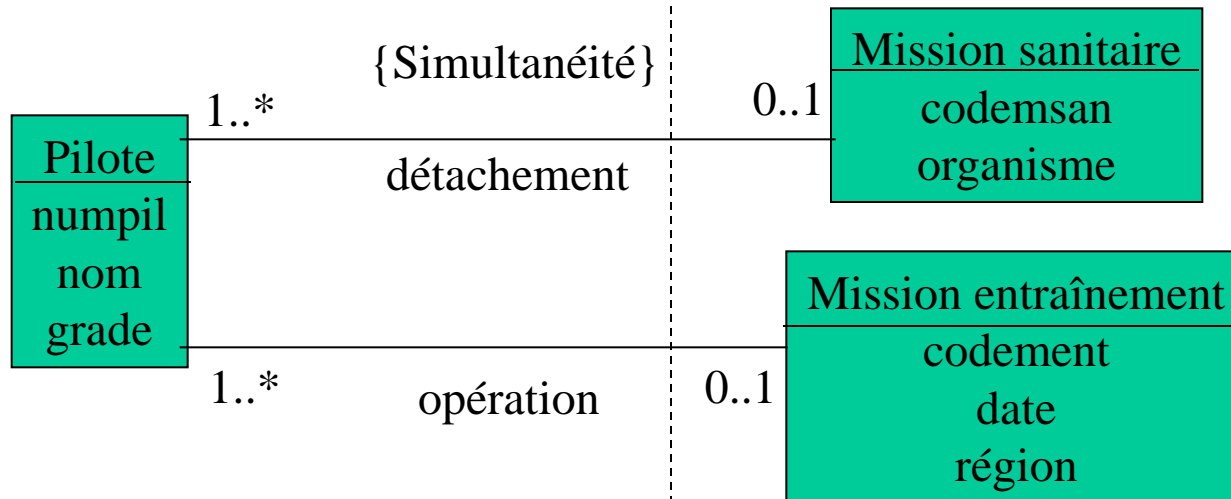


# Contrainte de simultanéité

Merise



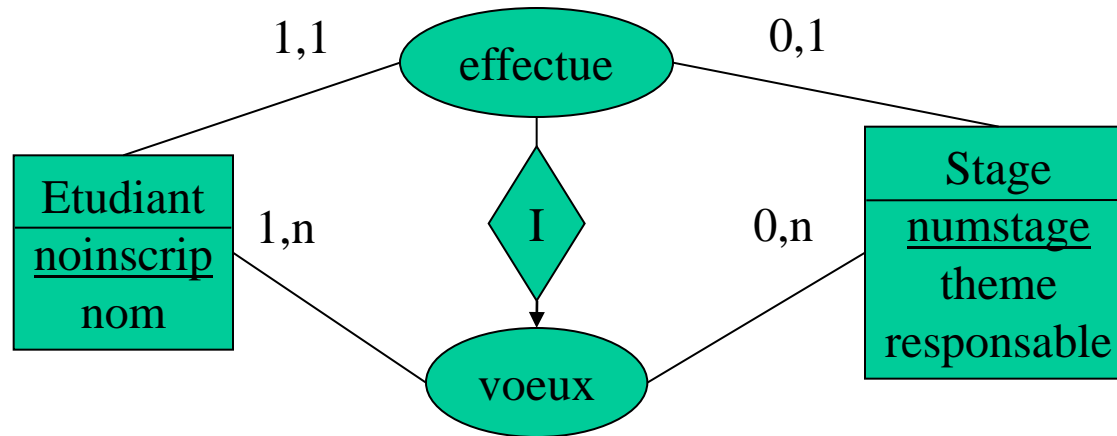
UML



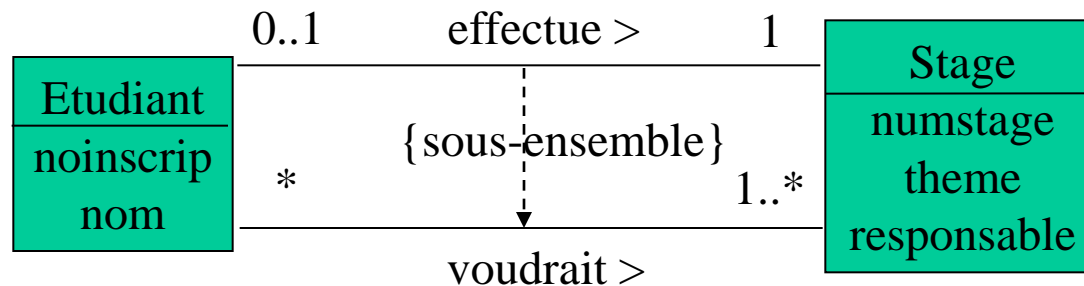


# Contrainte d'inclusion

Merise



UML

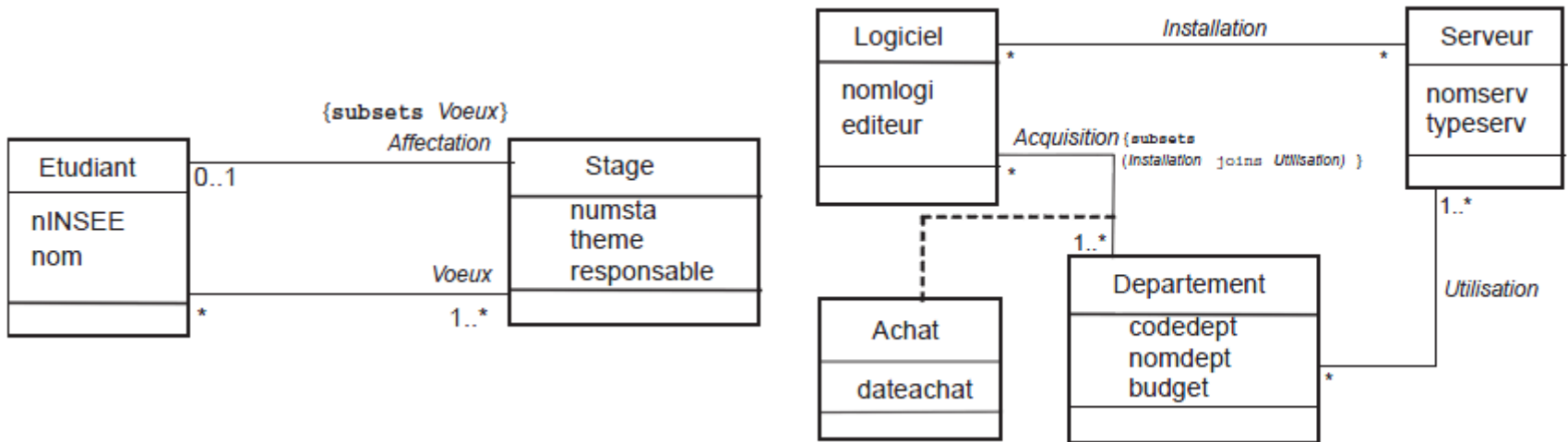


{ subsets *nom\_rôle* } est possible.

# Contrainte d'inclusion

## { subsets *nom\_rôle* }

- La spécification UML 2 propose la notation { subsets *nom\_rôle* } pour exprimer une contrainte d'inclusion.
- Alors que la spécification présente des exemples basés sur deux associations binaires, il est possible d'étendre cette notation à plus de deux associations et d'utiliser les classes-associations pour celles qui concernent les associations *n-aires*.



# *Représentation des contraintes*

Des contraintes peuvent être ajoutées aux attributs et opérations d'une classe

notation entre { }

**Invariants** = Propriétés vraies pour l'ensemble des instances de la classe

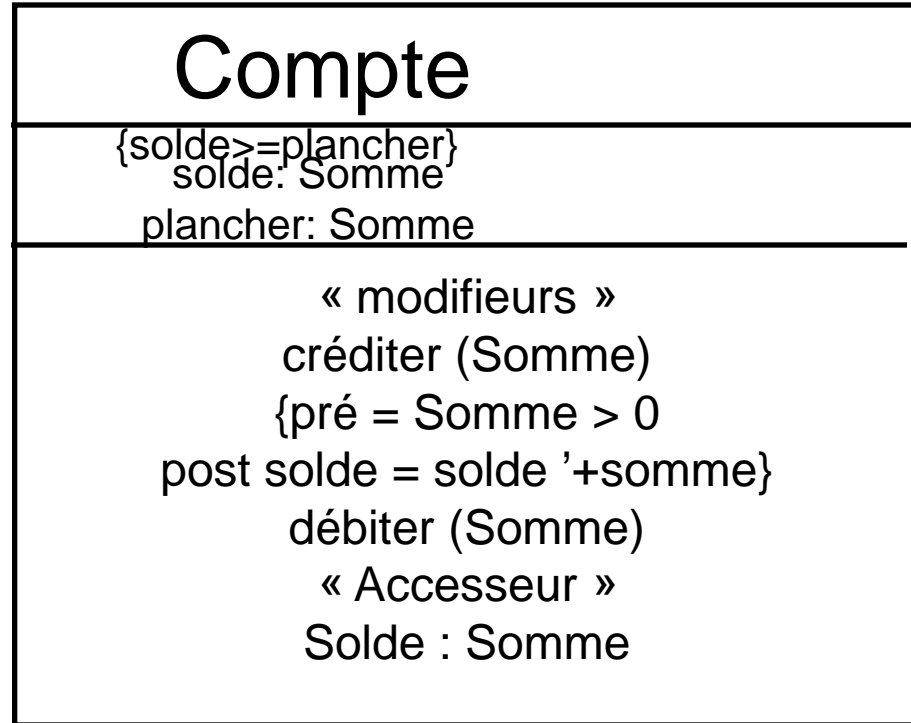
dans un état stable en dehors de l'exécution d'une méthode), chaque instance doit vérifier les invariants de sa classe

**Pré-condition** = Propriété devant être assurée par le client avant l'exécution de l'opération. Sans respect de cette pré-condition, l'opération ne garantit rien.

**Post-condition** : Propriété garantie par la méthode après son exécution.

**Contraintes sur un attribut** : cf OCL

# *Représentation des contraintes*



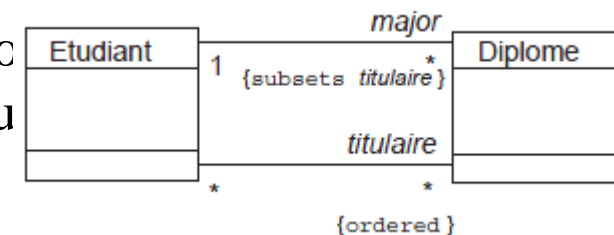
# Les contraintes prédéfinies de UML 2

La spécification UML 2 définit quelques contraintes qui se notent à l'extrémité d'un lien d'association (au même niveau que le rôle). Nous les citons ici pour mémoire mais elles ne sont pas toutes intéressantes dans un contexte de bases de données.

- {subsets *nom\_rôle*} exprime l'inclusion d'une association par rapport à une autre.
- {redefined *nom\_rôle*} redéfinit un rôle.
- {union} signifie que le rôle rassemble une union de sous-ensembles.
- {ordered} exprime un ordre au niveau des objets reliés.
- {bag} exprime qu'un même objet cible peut apparaître plusieurs fois dans l'assoc.
- {sequence} notée aussi {seq} combine les caractéristiques de *bag* et de *ordered*.
- {xor} qui indique le *ou exclusif* entre objets reliés par deux associations.

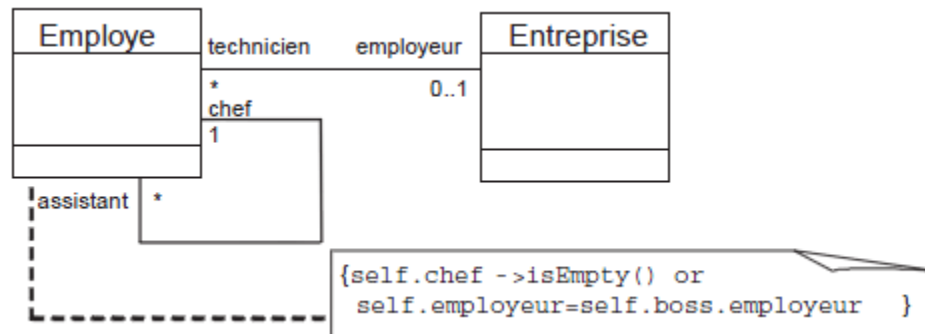
En l'absence de ces contraintes (cas le plus général et celui par défaut), la terminaison d'une association est dite de type *set* (*chaque objet relié n'est présent au plus qu'une seule fois dans l'association*).

*ordered* précise l'ordre d'obtention d'un diplôme (chronologique par exemple), alors que *subsets* indique que, pour être major d'un diplôme, l'étudiant doit aussi y être associé en tant que titulaire.



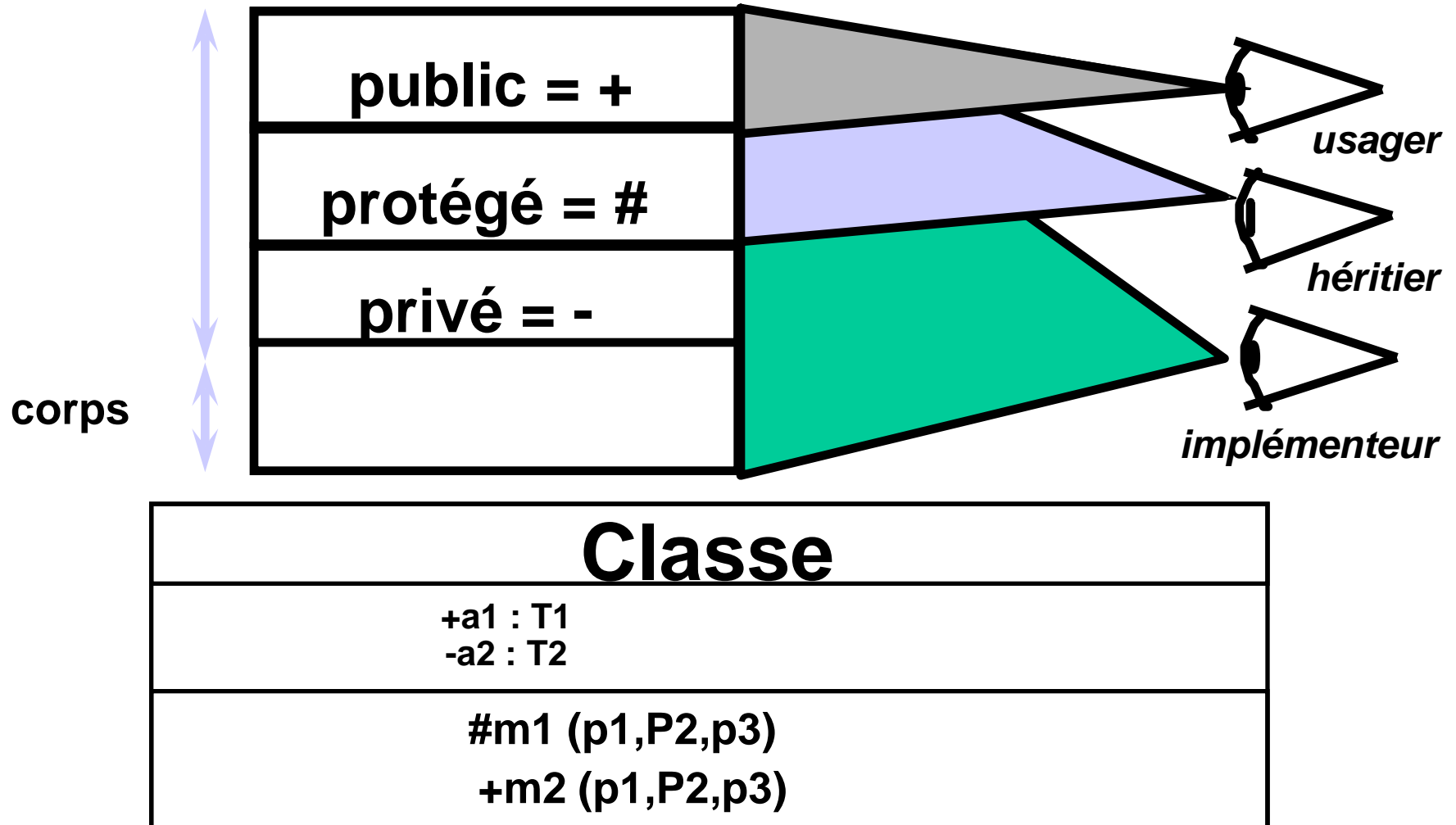
# Contrainte personnalisée avec OCL

- Pour toute autre contrainte, OCL est utilisable dans une note.  
L'exemple décrit la contrainte qui exprime, à l'aide du langage OCL, que tout employé est soit :
  - chef, et dans ce cas il n'a pas de chef (condition `isEmpty` vérifiée) ;
  - sous la responsabilité d'un chef qui doit être employé dans la même entreprise que lui (deuxième partie de la condition).



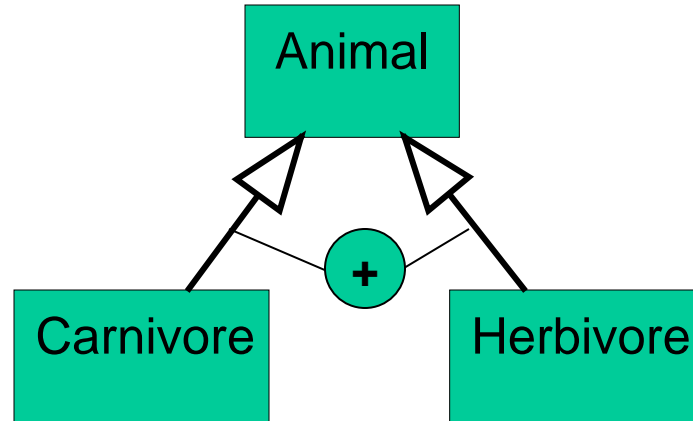
- Toute contrainte prédéfinie par UML2 ou exprimée à l'aide du langage OCL sera à programmer par la suite sous SQL soit par contrainte de vérification (CHECK), soit par déclencheur.

## 6. Visibilité et Encapsulation : vers de bons objets



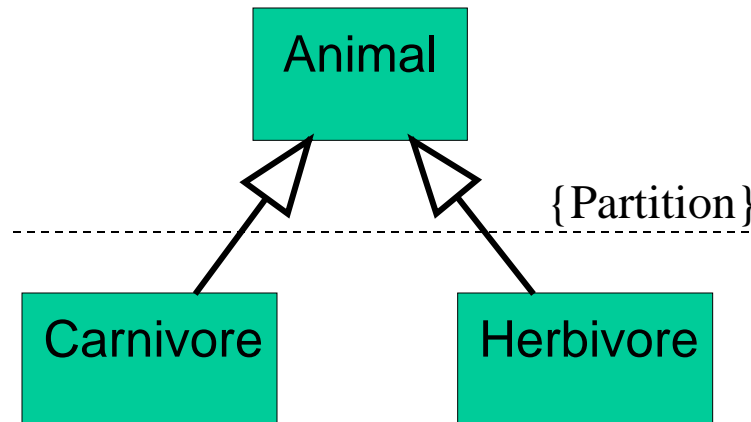
# Héritage

Merise



La notation est équivalente pour les autres formes d'héritage, à savoir exclusion et totalité.

UML



Le cas sans contrainte est représenté sans ornement particulier





## Chapitre 6

# Modèle Logique des Données relationnelles

Michel Dubois

I.U.T. de Vannes

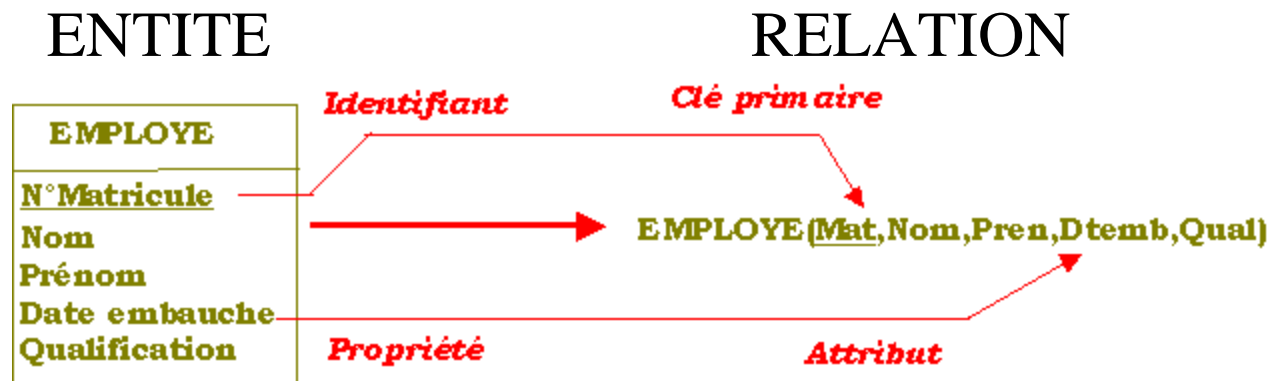
[Michel.Dubois@univ-ubs.fr](mailto:Michel.Dubois@univ-ubs.fr)

[Retour au plan](#)

# RÈGLES DE PASSAGE DEPUIS UN MODÈLE ENTITÉ-ASSOCIATION VERS UN SCHÉMA RELATIONNEL

REGLE n°1 : TOUTE ENTITE DEVIENT UNE  
RELATION dans laquelle :

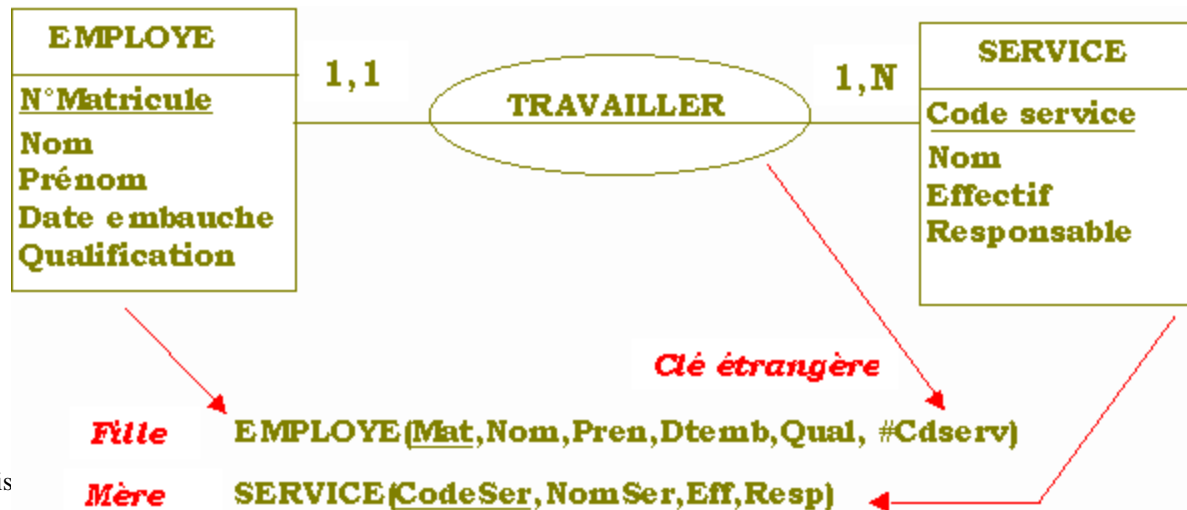
- les attributs traduisent les propriétés de l'entité
- la clé primaire traduit l'identifiant de l'entité



# RÈGLES DE PASSAGE DEPUIS UN MODÈLE ENTITÉ-ASSOCIATION VERS UN SCHÉMA RELATIONNEL

## REGLE n°2 : 1:N

Une association de dimension 2 avec cardinalité maximale à 1 se réécrit en portant dans la relation fille la clé primaire de la relation mère. il s 'agit d 'une clé étrangère. Par convention, nous la symboliserons au moyen de #.





# *RÈGLES DE PASSAGE DEPUIS UN MODÈLE ENTITÉ-ASSOCIATION VERS UN SCHÉMA RELATIONNEL*

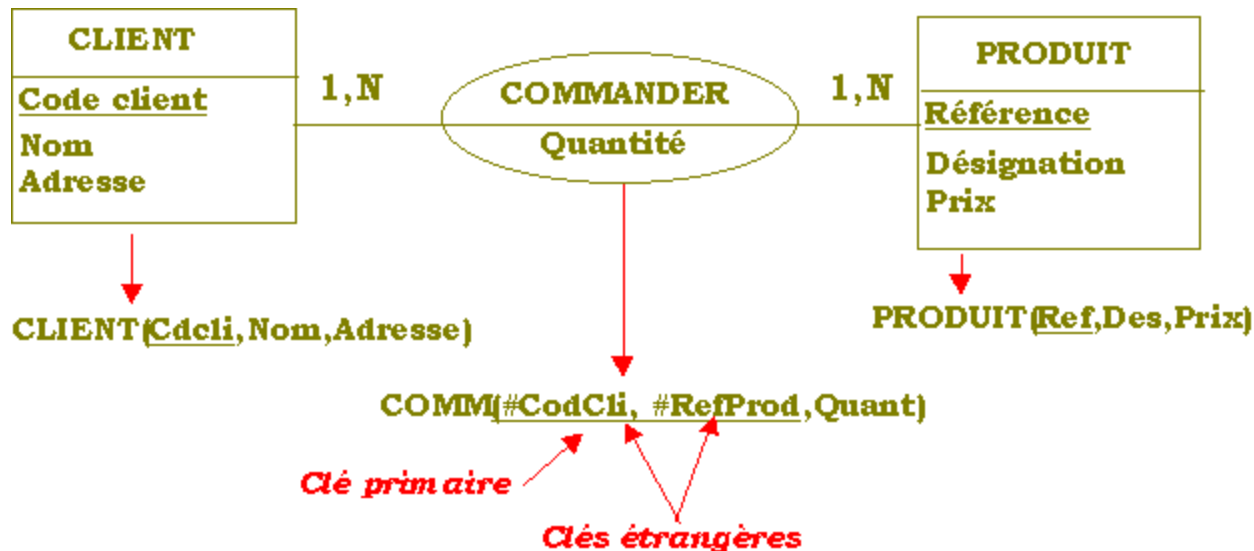
REGLE n°3 N:M (cas de la dimension 2 plusieurs à plusieurs) :

Une association de dimension 2 avec cardinalités plusieurs à plusieurs se réécrit en :

- créant une relation particulière qui contient comme attributs les identifiants des 2 entités associées
- ces attributs constituent à eux 2 la clé primaire de la relation
- ils sont individuellement clés étrangères
- ajoutant la ou les éventuelles propriétés de l'association à cette relation.

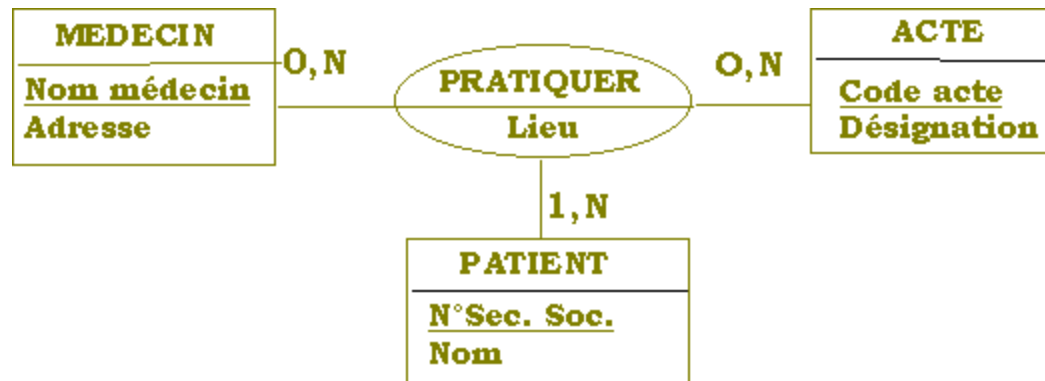
# RÈGLES DE PASSAGE DEPUIS UN MODÈLE ENTITÉ-ASSOCIATION VERS UN SCHÉMA RELATIONNEL

REGLE n°3 N:M (cas de la dimension 2 plusieurs à plusieurs) :



# *RÈGLES DE PASSAGE DEPUIS UN MODÈLE ENTITÉ-ASSOCIATION VERS UN SCHÉMA RELATIONNEL*

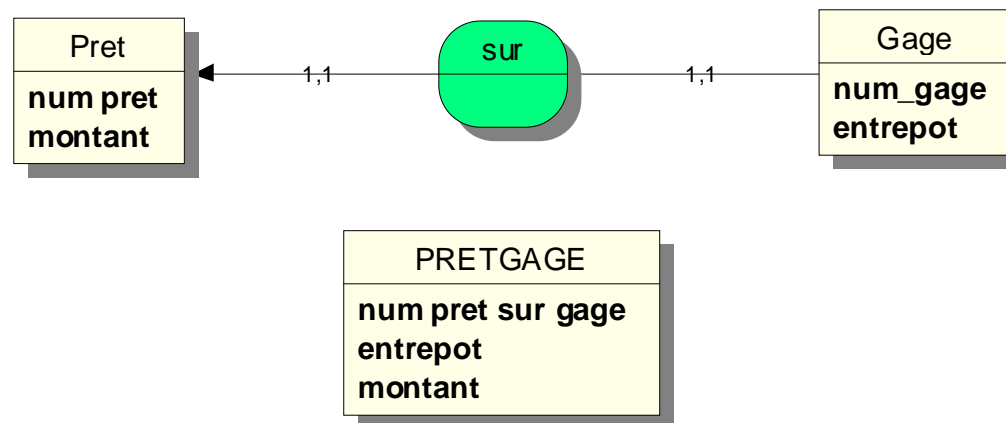
REGLE n°3 N:M (cas de la dimension 3 non décomposable) :



**MEDECIN**(NomMed, Adr)  
**ACTE**(CdActe, Des)  
**PATIENT**(SS, NomPat)  
**PRATIQUER**(#Med, #Acte, #Secu, Lieu)

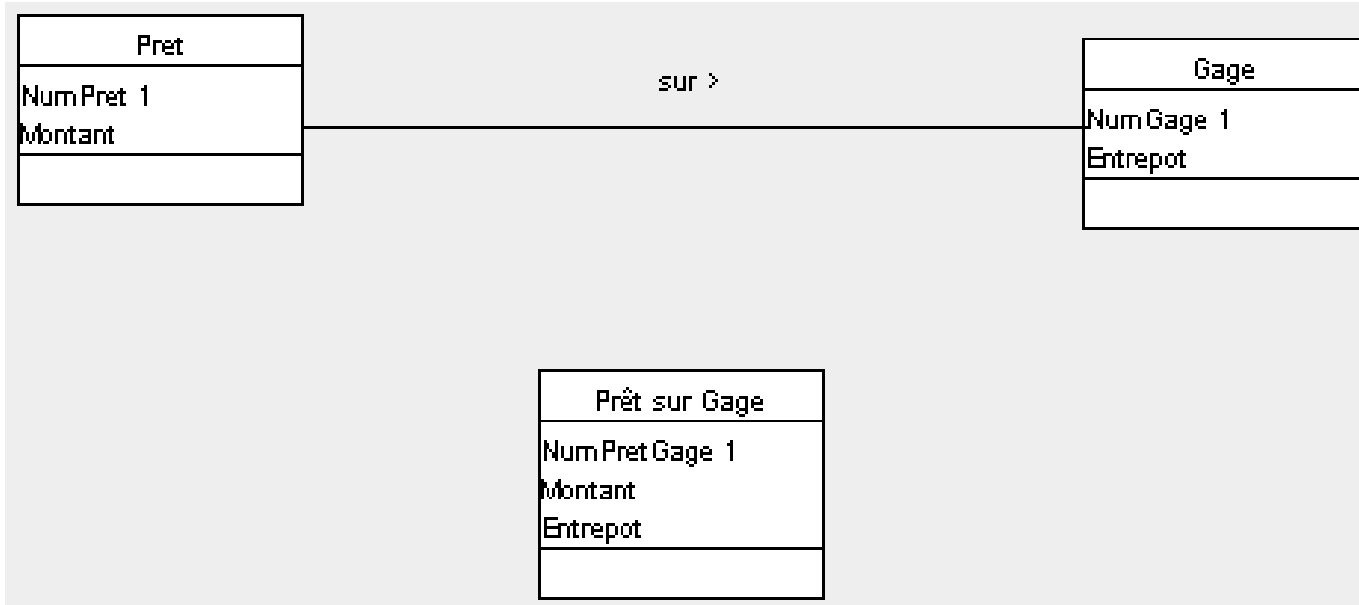
## *ASSOCIATION 1:1 $\rightarrow$ (1,1)-(1,1)*

- Une telle association peut se réduire à une seule entité.
- La règle n°1 s'applique. Des vues peuvent reconstituer les deux relations.



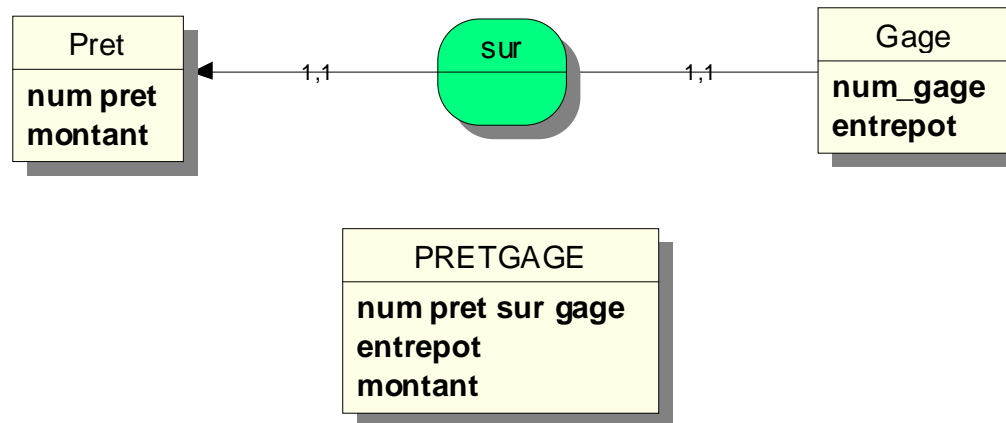
# *ASSOCIATION 1:1 -> (1)-(1)*

- Une telle association peut se réduire à une seule classe.
- La règle n°1 s'applique. Des vues peuvent reconstituer les deux relations.

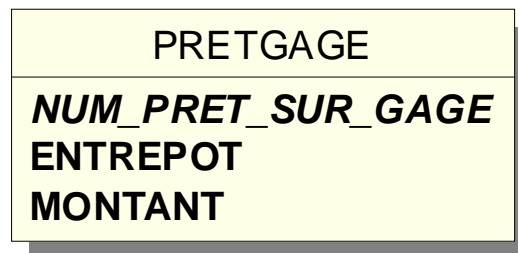




# *ASSOCIATION 1:1 -> (1,1)-(1,1)*

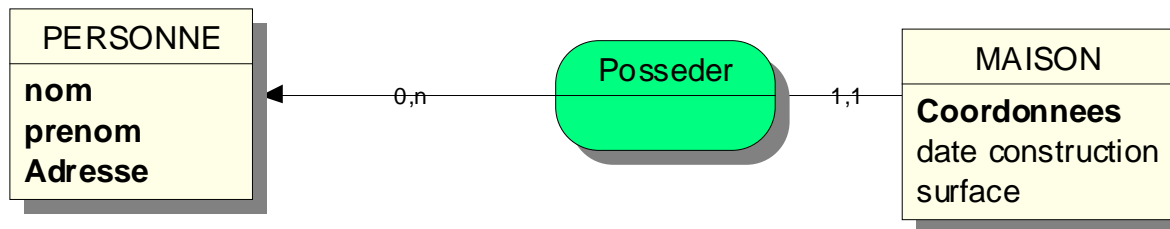


Toutes les df sont intra-relations et ne posent pas de problème de contraintes supplémentaires.



# *ASSOCIATION 1:N -> (1,1)-(0,N)*

- La règle n°2 s'applique.

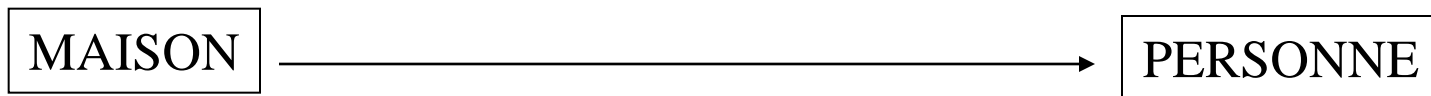
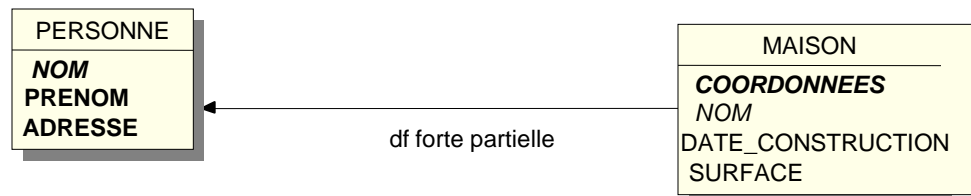
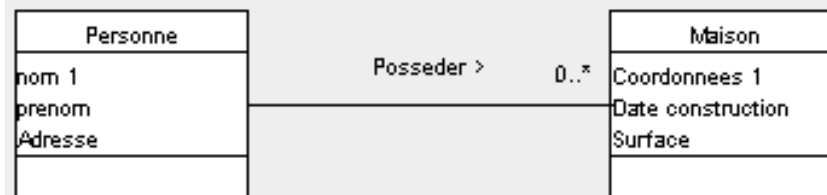
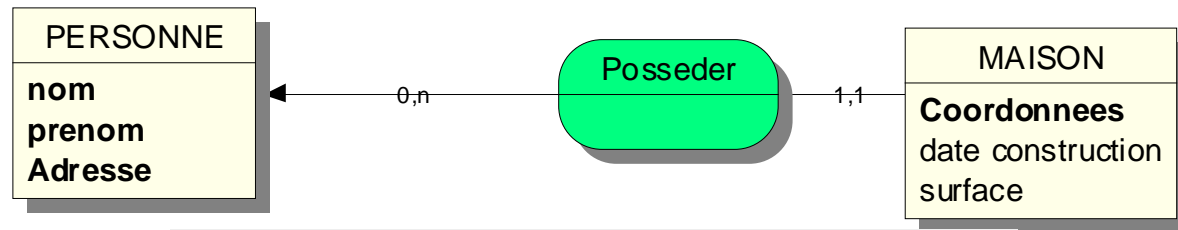


## *ASSOCIATION 1:N -> (1)-(0,\*)*

- La règle n°2 s'applique.



# ASSOCIATION 1:N -> (1,1)-(0,N)



MAISON.#NOM est obligatoire

MAISON.#NOM est unique

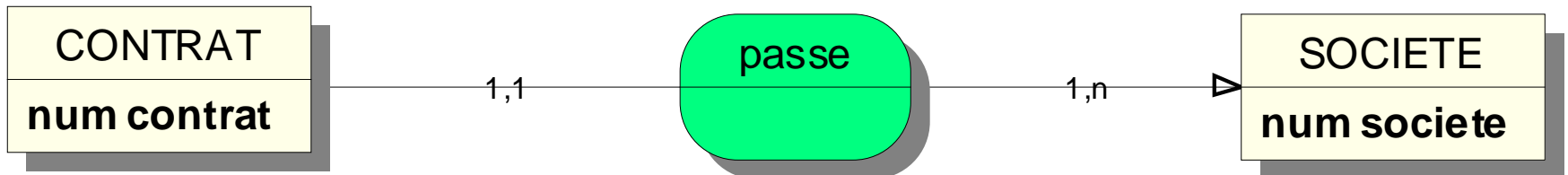
# ASSOCIATION 1:N -> (1,1)-(1,N)

- La règle n°2 s'applique.
- Une contrainte de dépendance fonctionnelle totale :  
CONTRAT.num\_contrat ->(T) SOCIETE.num\_societe  
Toute société est associé à un contrat. Donc l'ensemble des sociétés présent dans la table contrat est égal à l'ensemble des sociétés de la table société.

CONTRAT[num\_societe]=SOCIETE[num\_societe]      <=>

CONTRAT[num\_societe]⊆SOCIETE[num\_societe]: **clé étrangère**

et CONTRAT[num\_societe]⊇SOCIETE[num\_societe]



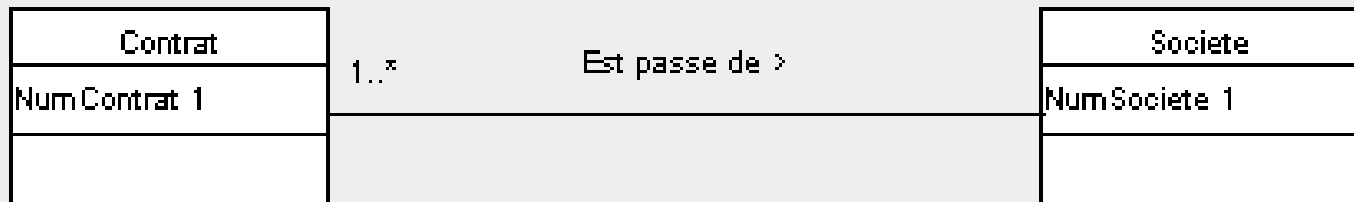
# ASSOCIATION 1:N -> (1,1)-(1,\*)

- La règle n°2 s'applique.
- Une contrainte de dépendance fonctionnelle totale :  
 $\text{CONTRAT.NumContrat} \rightarrow (\text{T}) \text{SOCIETE.NumSociete}$   
Toute société est associée à un contrat. Donc l'ensemble des sociétés présent dans la table contrat est égal à l'ensemble des sociétés de la table société.

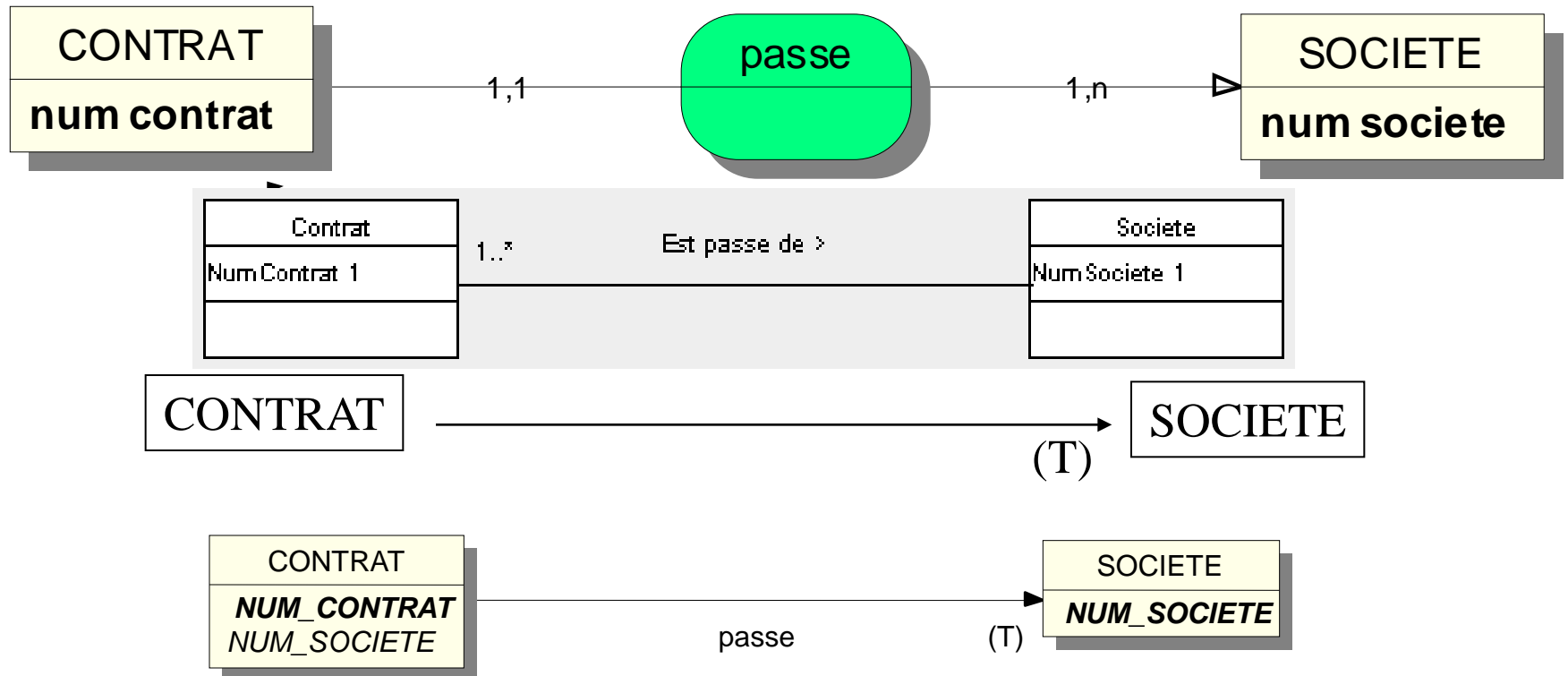
$\text{CONTRAT}[\text{NumSociete}] = \text{SOCIETE}[\text{NumSociete}] \quad \Leftrightarrow$

$\text{CONTRAT}[\text{NumSociete}] \subseteq \text{SOCIETE}[\text{NumSociete}]$  : **clé étrangère**

et  $\text{CONTRAT}[\text{NumSociete}] \supseteq \text{SOCIETE}[\text{NumSociete}]$



# ASSOCIATION 1:N -> (1,1)-(1,N)



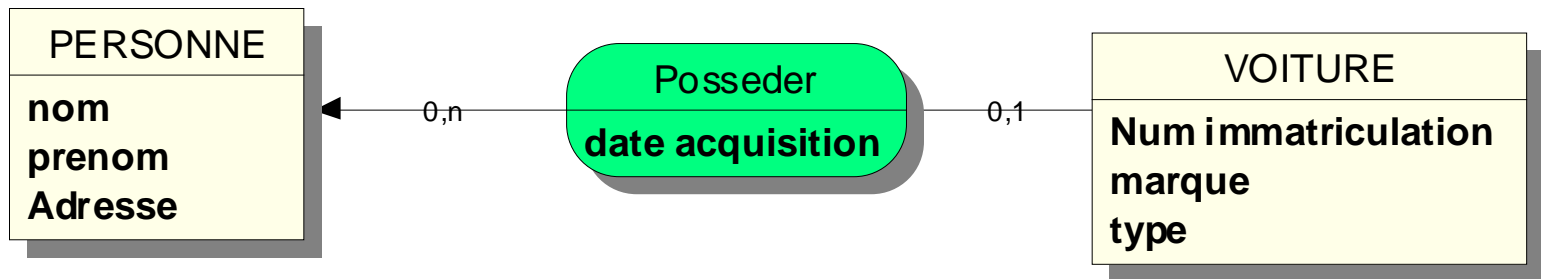
CONTRAT.num\_societe est obligatoire

CONTRAT.num\_societe est unique

$SOCIETE[num\_societe] \subseteq CONTRAT[num\_societe]$

## *ASSOCIATION 1:N -> (0,1)-(0,N)*

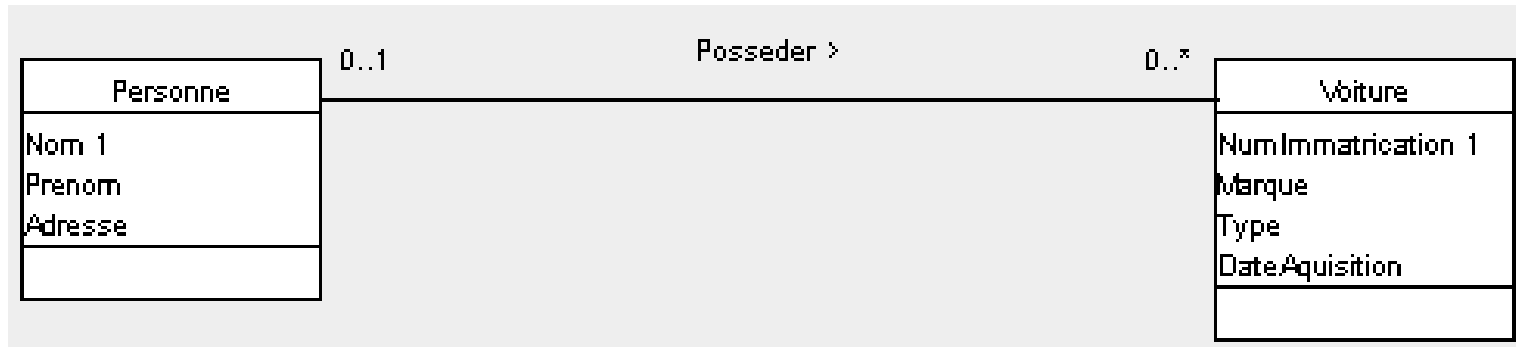
- La règle n°2 s'applique.



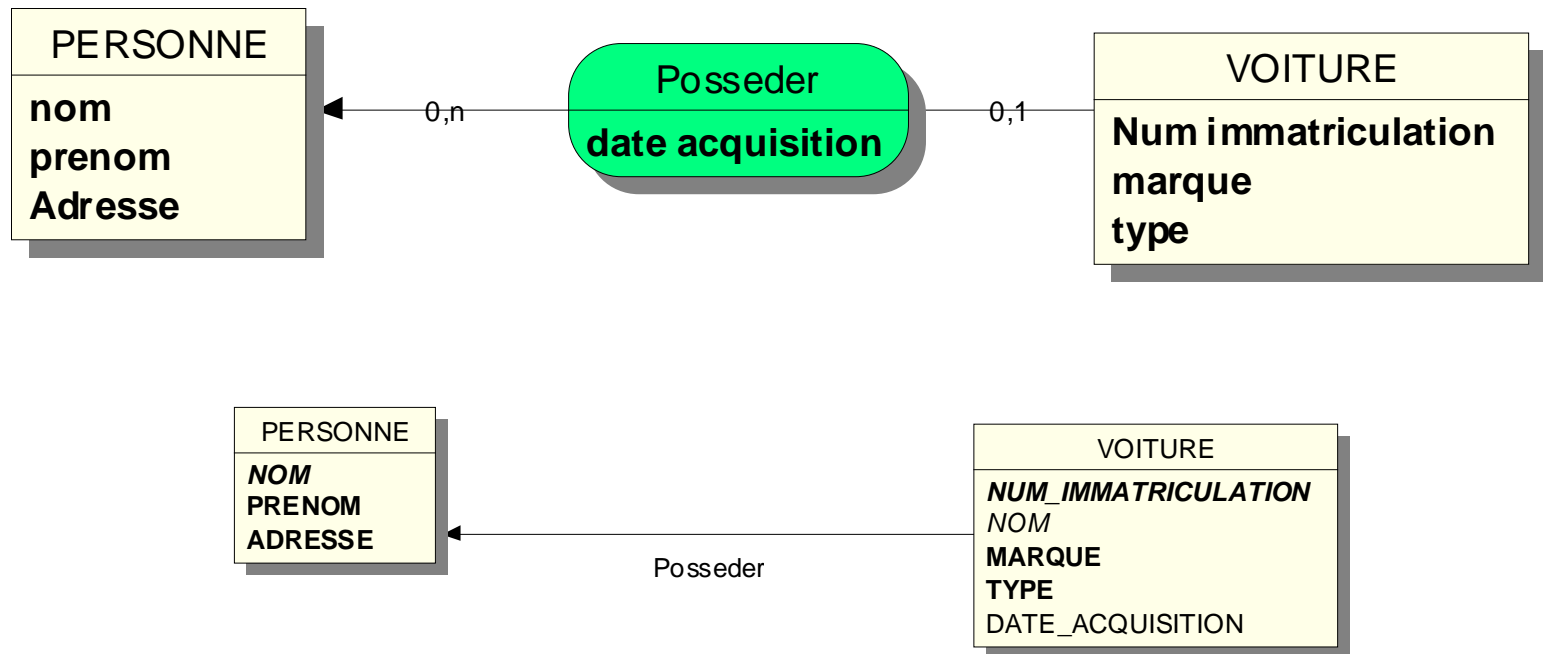


# *ASSOCIATION 1:N -> (0,1)-(0,\*)*

- La règle n°2 s'applique.

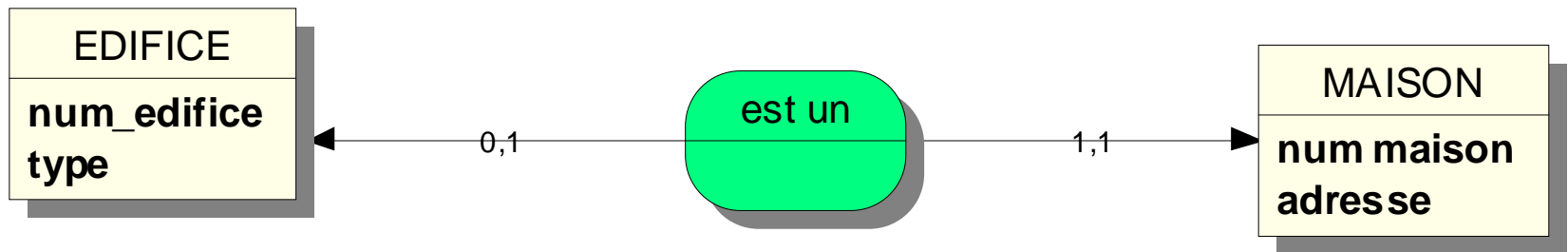


# ASSOCIATION 1:N -> (0,1)-(0,N)

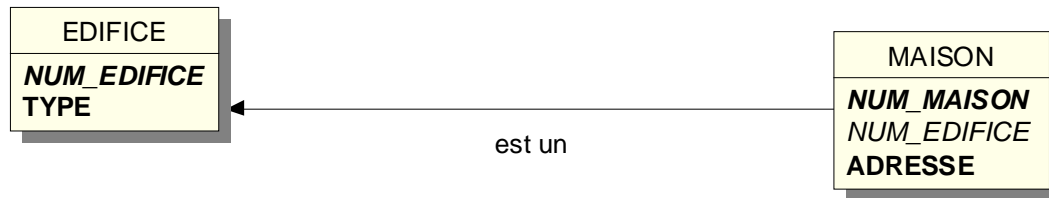
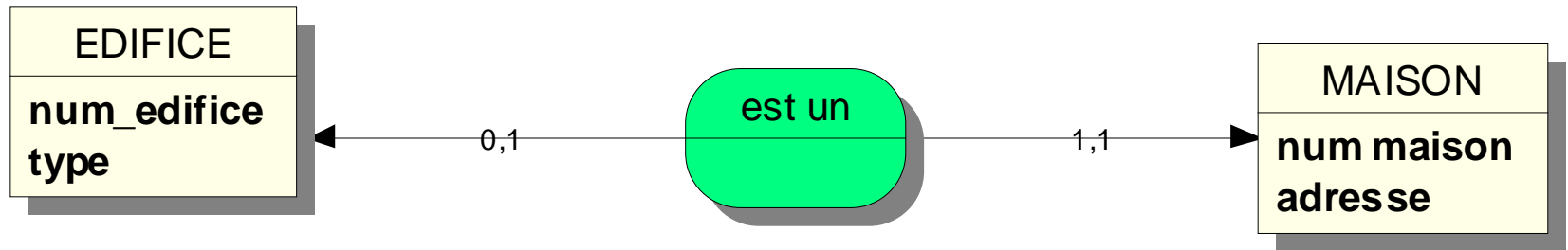


## *ASSOCIATION 1:1 -> (1,1)-(0,1)*

- La règle n°2 s 'applique qu 'une seule fois.
- On va privilégier la dépendance fonctionnelle forte.



# *ASSOCIATION 1:1 -> (1,1)-(0,1)*

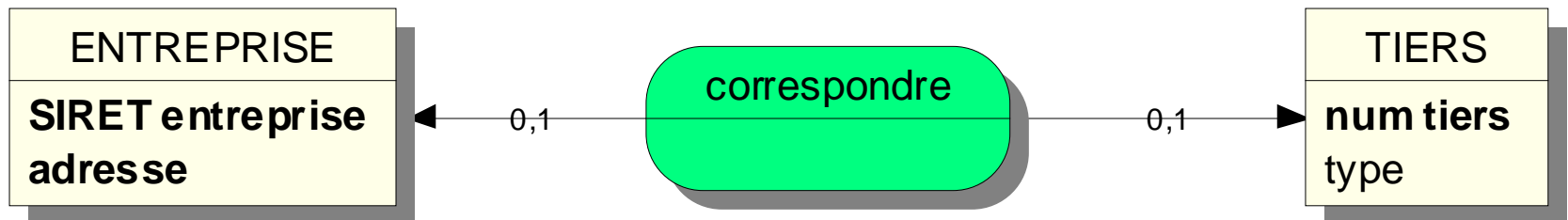


MAISON.NUM\_EDIFICE est obligatoire.

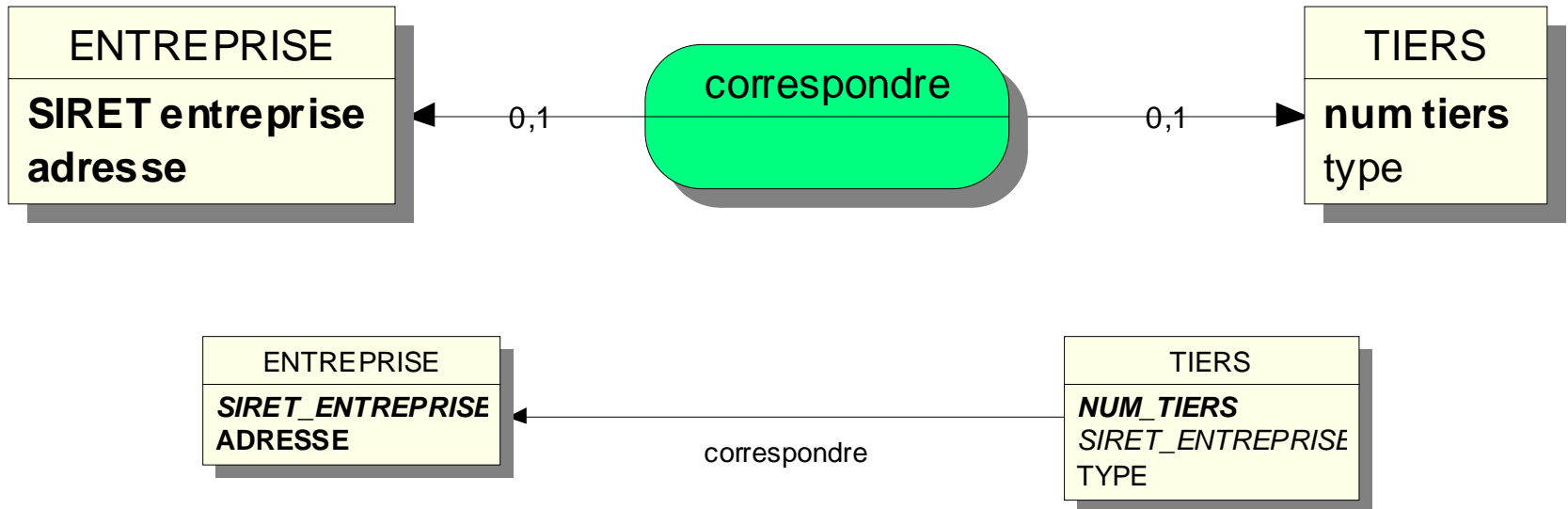
MAISON.NUM\_EDIFICE n 'accepte pas de doublon.

## *ASSOCIATION 1:1 -> (0,1)-(0,1)*

- La règle n°2 s 'applique qu 'une seule fois.
- On va privilégier un sens (au choix)



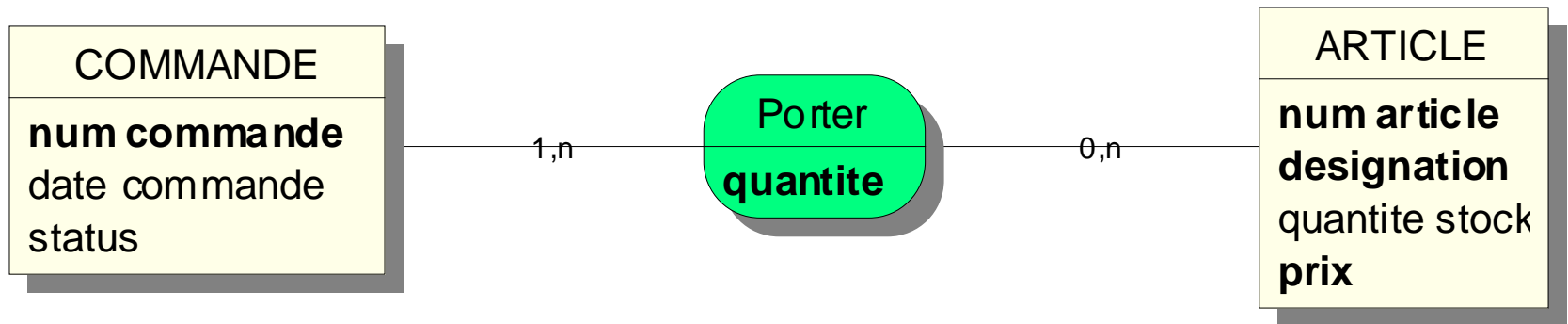
# *ASSOCIATION 1:1 -> (0,1)-(0,1)*



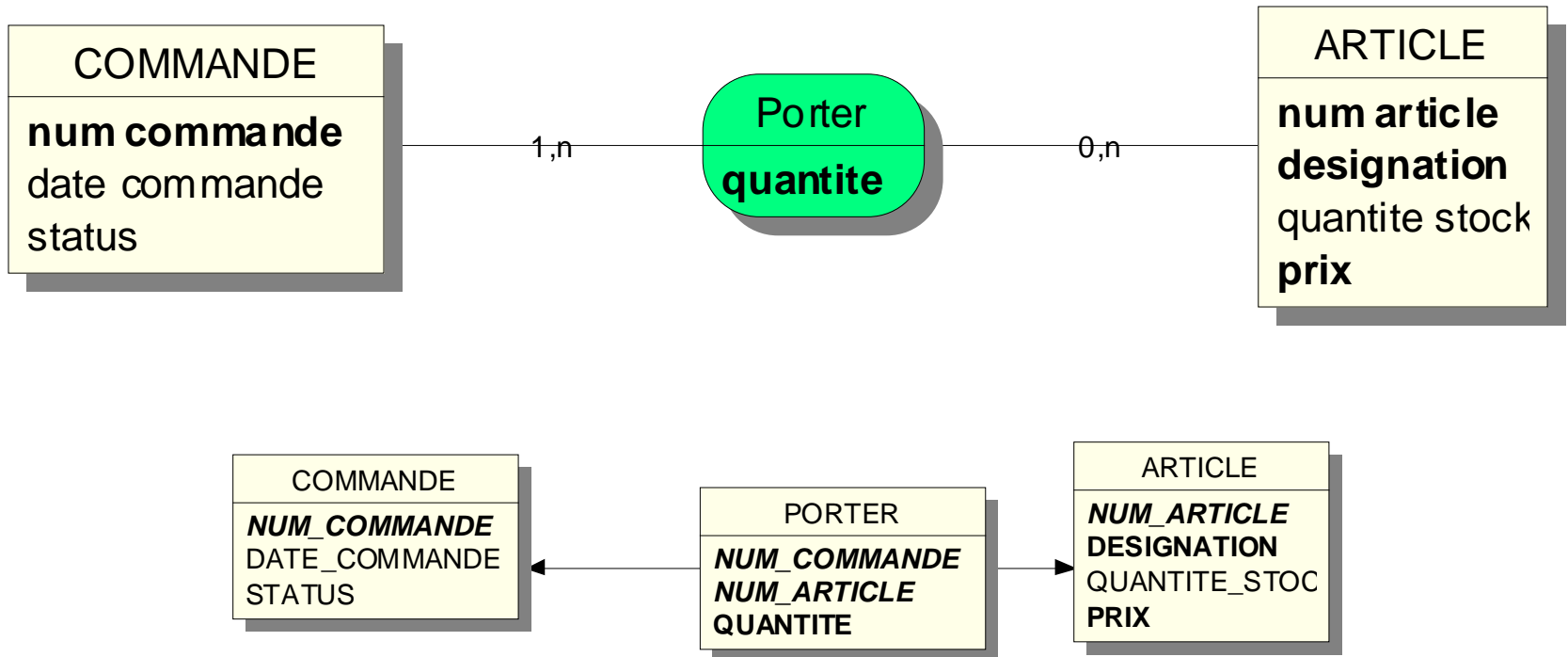
TIERS.SIRET\_ENTREPRISE n 'accepte pas de doublon

# *ASSOCIATION N:M -> (X,N)-(X,N)*

- La règle n°3 s'applique.



# ASSOCIATION $N:M \rightarrow (X,N)-(X,N)$





# *ASSOCIATION $N:M \rightarrow (1,N)-(0,N)$*

- La règle n°3 s'applique.
- Mais il y a aussi une contrainte de dépendance fonctionnelle totale :
  - $\text{Porter.num\_commande} \rightarrow (\text{T})\text{Commande.num\_commande}$
  - Toute commande de la relation commande se retrouve dans la relation porter.

$\text{COMMANDE}[\text{num\_commande}] = \text{PORTER}[\text{num\_commande}]$

$\Leftrightarrow \text{PORTER}[\text{num\_commande}] \subseteq \text{COMMANDE}[\text{num\_commande}] : \text{FK}$

et  $\text{COMMANDE}[\text{num\_commande}] \subseteq \text{PORTER}[\text{num\_commande}]$

On teste :

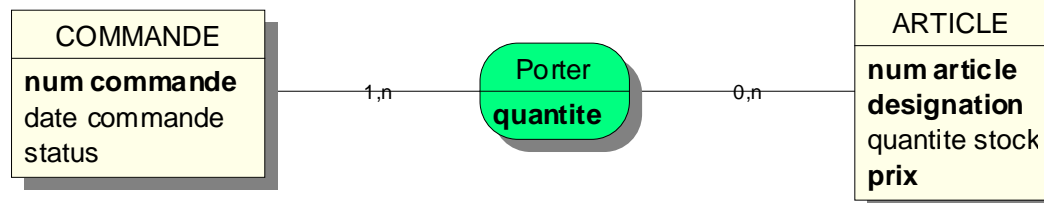
$\text{COMMANDE}[\text{num\_commande}] - \text{PORTER}[\text{num\_commande}] = \emptyset$

...WHERE NOT EXISTS (

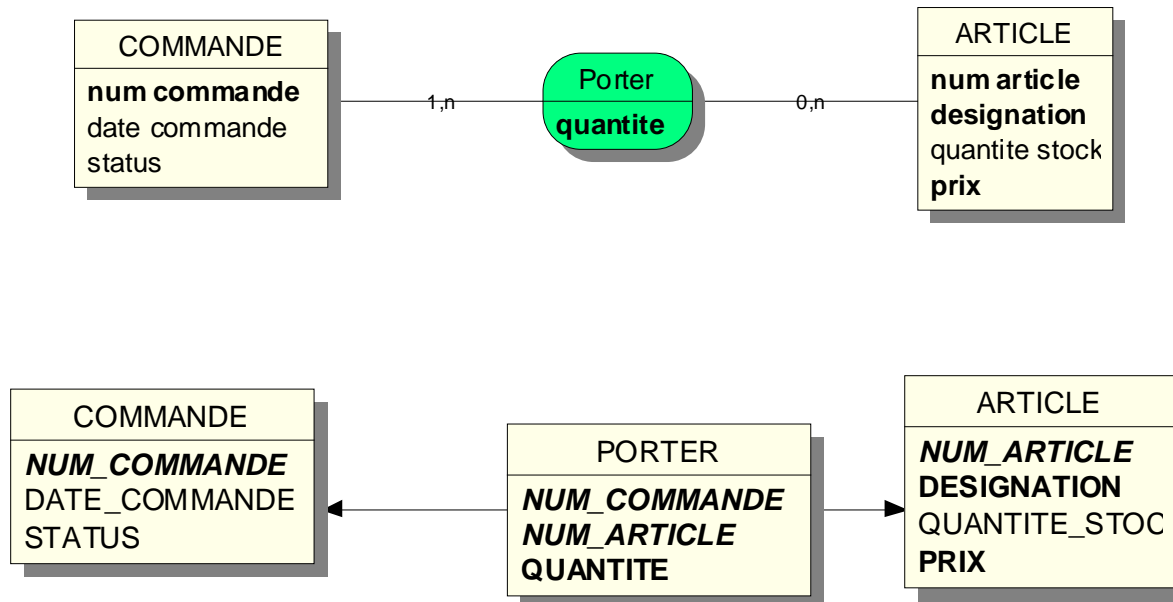
SELECT num\_commande FROM commande

MINUS

SELECT num\_commande FROM porter)

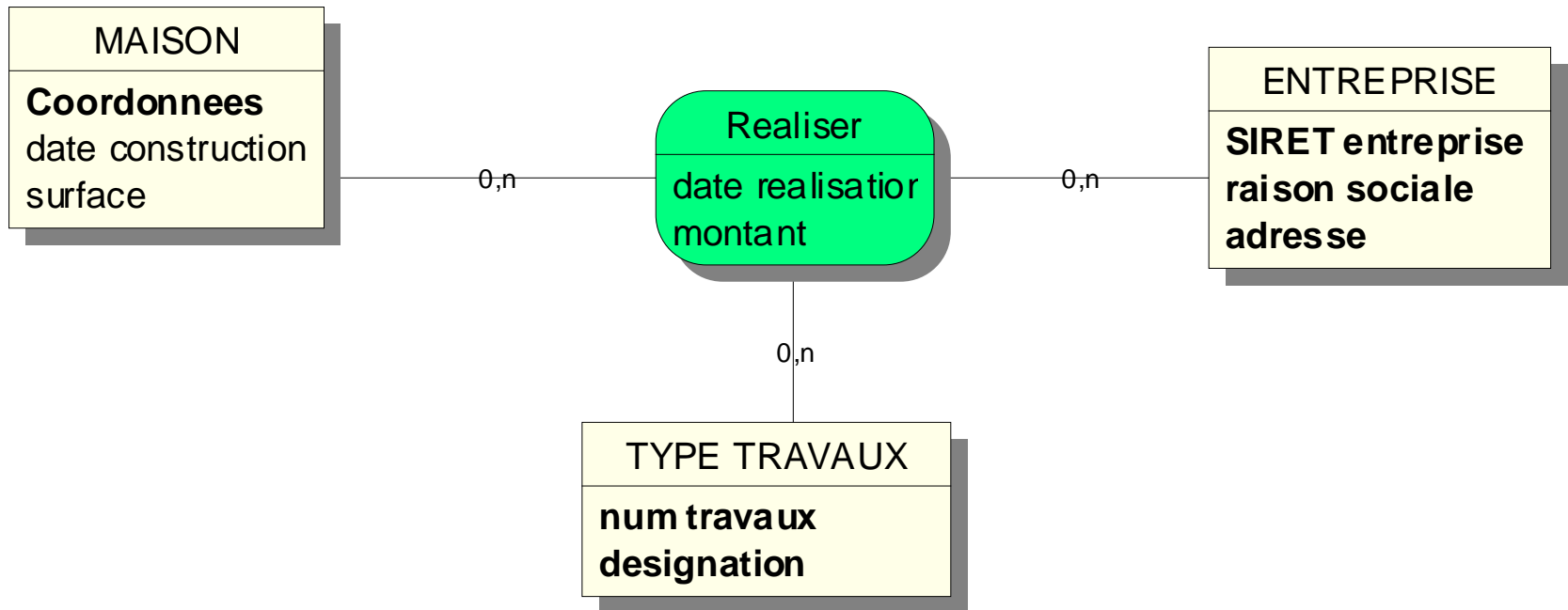


# ASSOCIATION $N:M \rightarrow (1,N)-(0,N)$

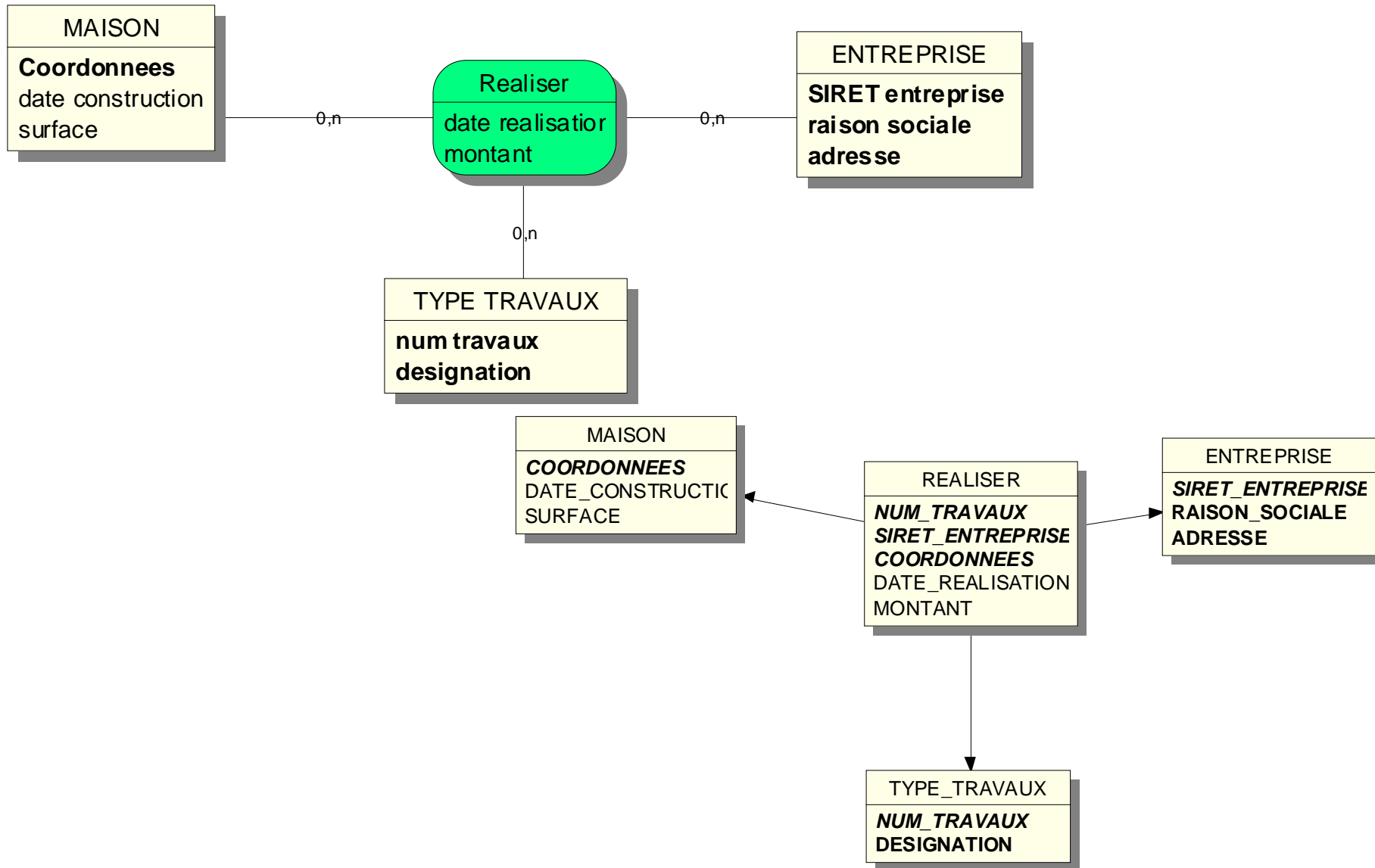


# ASSOCIATION N-AIRE

- La règle n°3 s'applique.



# ASSOCIATION N-AIRE



# ASSOCIATIONS LIÉES

## AUX CLASSES-ASSOCIATIONS UML

- Pour la traduction relationnelle des associations connectées à des classes-associations UML, il faut utiliser, pour chaque association, une des solutions précédemment étudiées en fonction de la nature de l'association (*plusieurs-à-un*, *un-à-plusieurs*, *plusieurs-à-plusieurs*).

Figure 3-31 Classe-association un-à-plusieurs

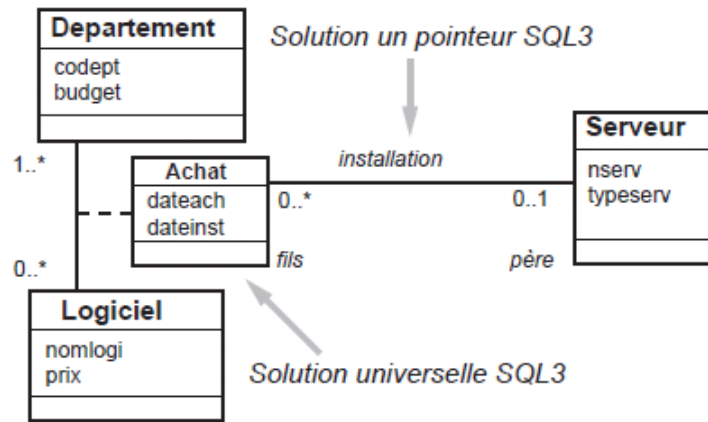


Figure 3-33 Classe-association plusieurs-à-plusieurs

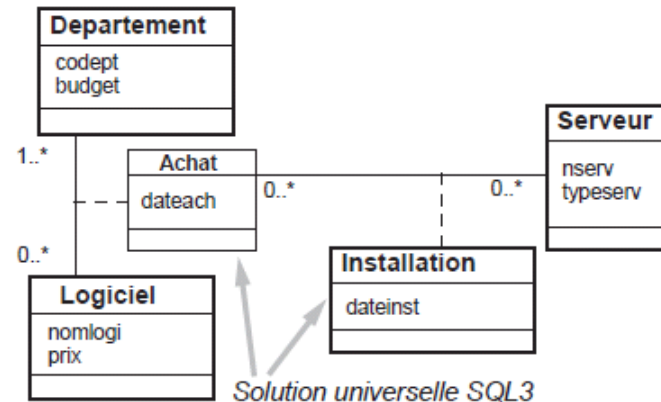
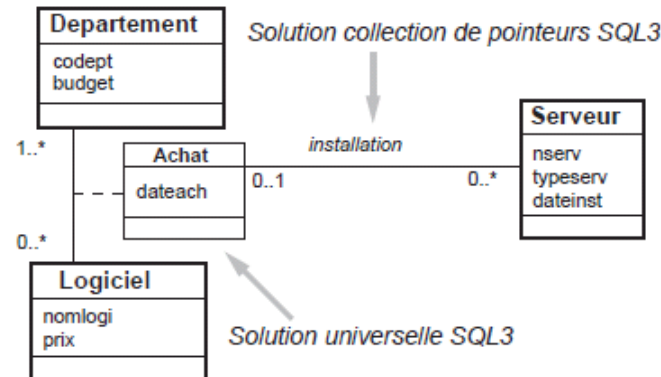
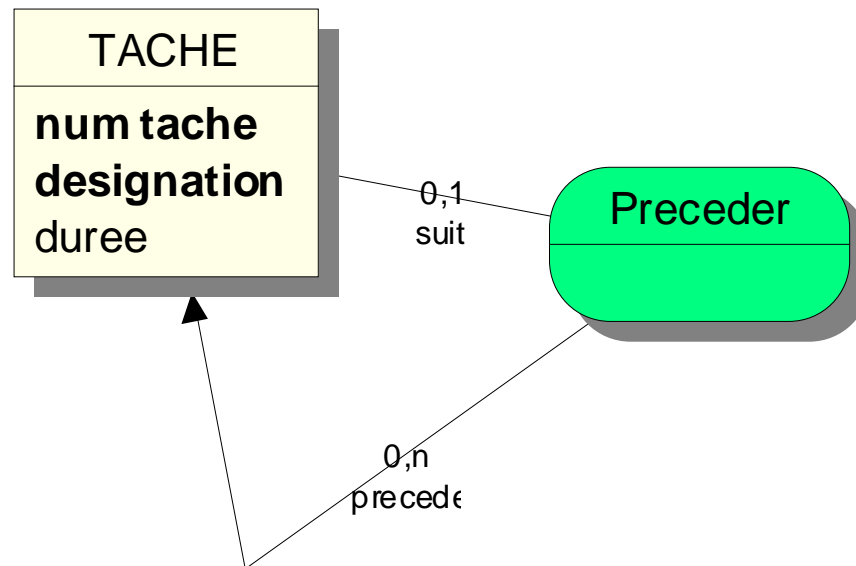


Figure 3-32 Classe-association plusieurs-à-un

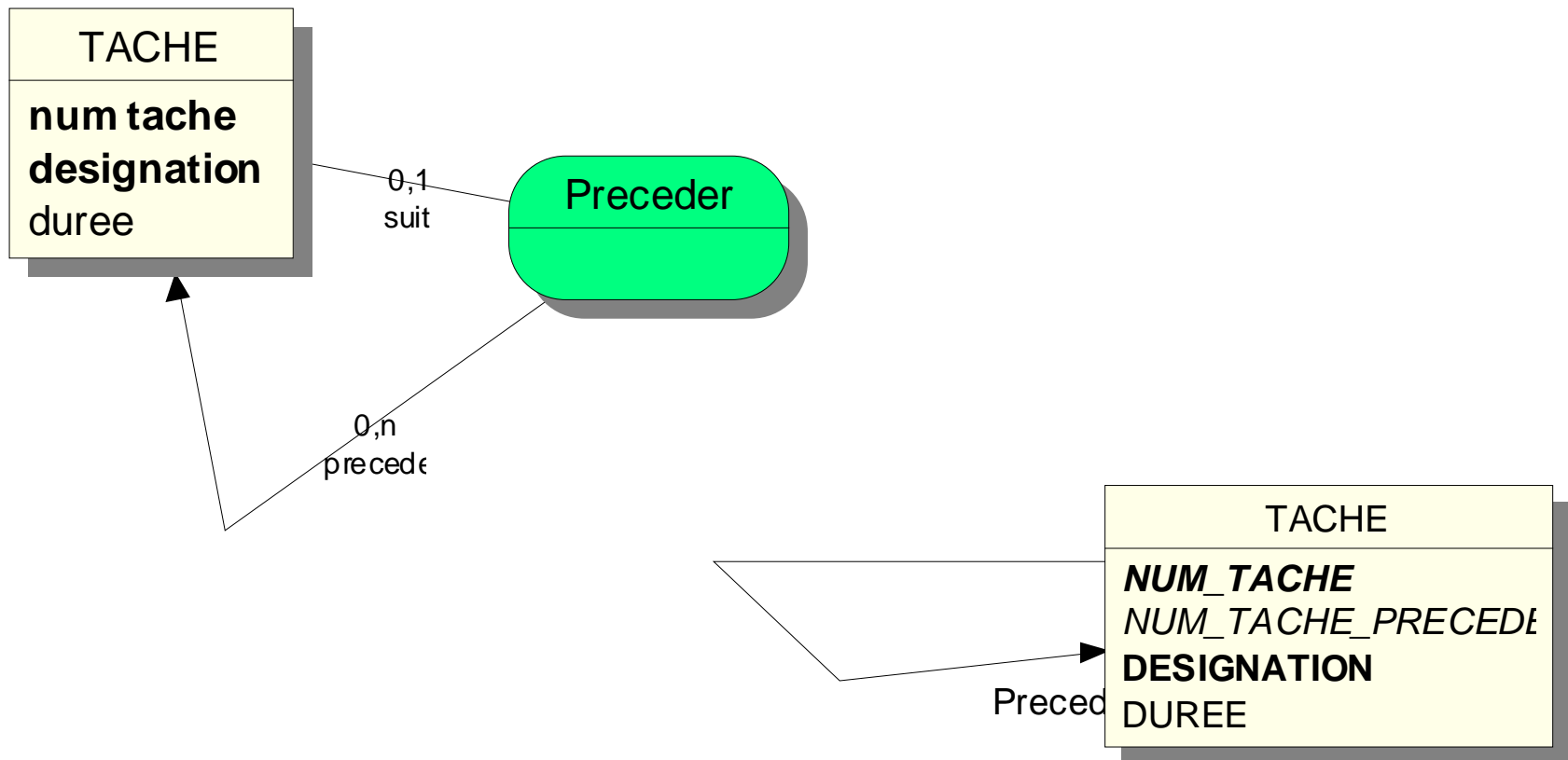


# *ASSOCIATION 1:N REFLEXIVE*

- La règle n°2 s 'applique.

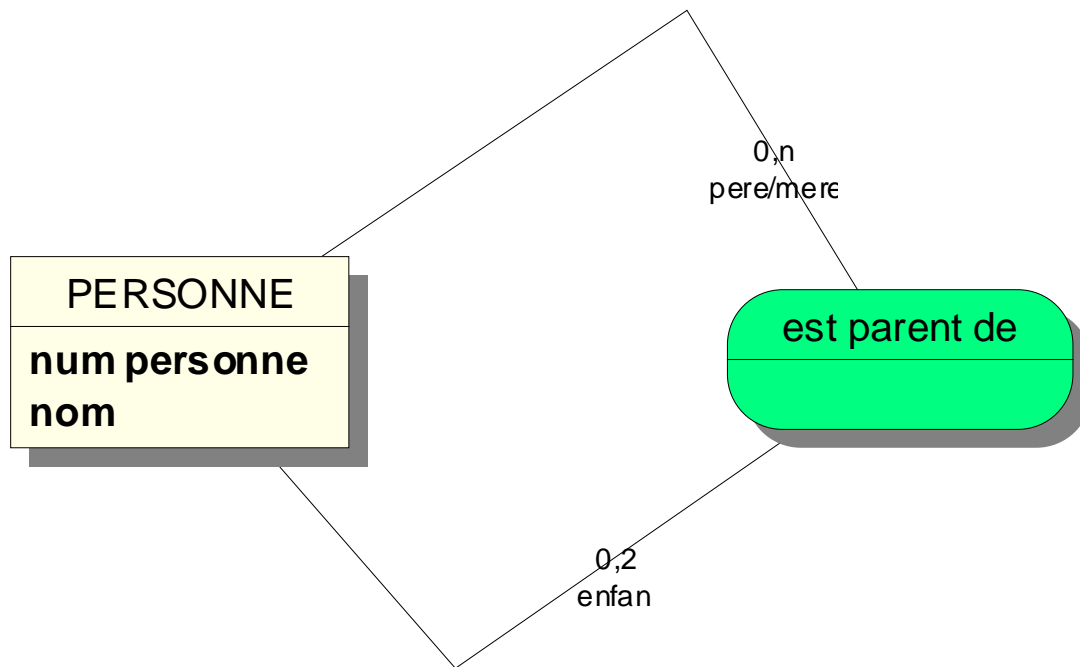


# ASSOCIATION 1:N REFLEXIVE



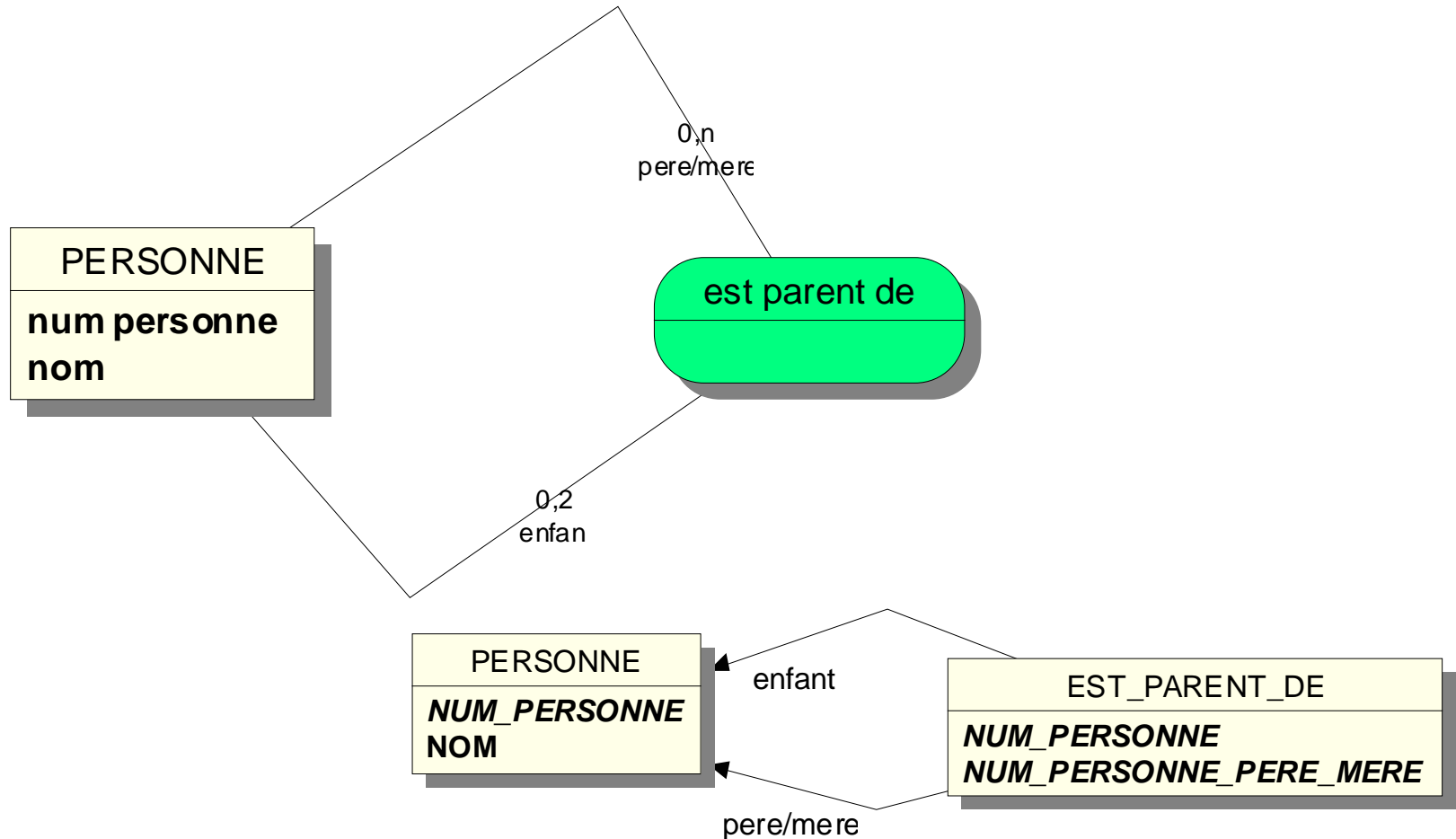
# *ASSOCIATION N:M REFLEXIVE*

- La règle n°3 s'applique.

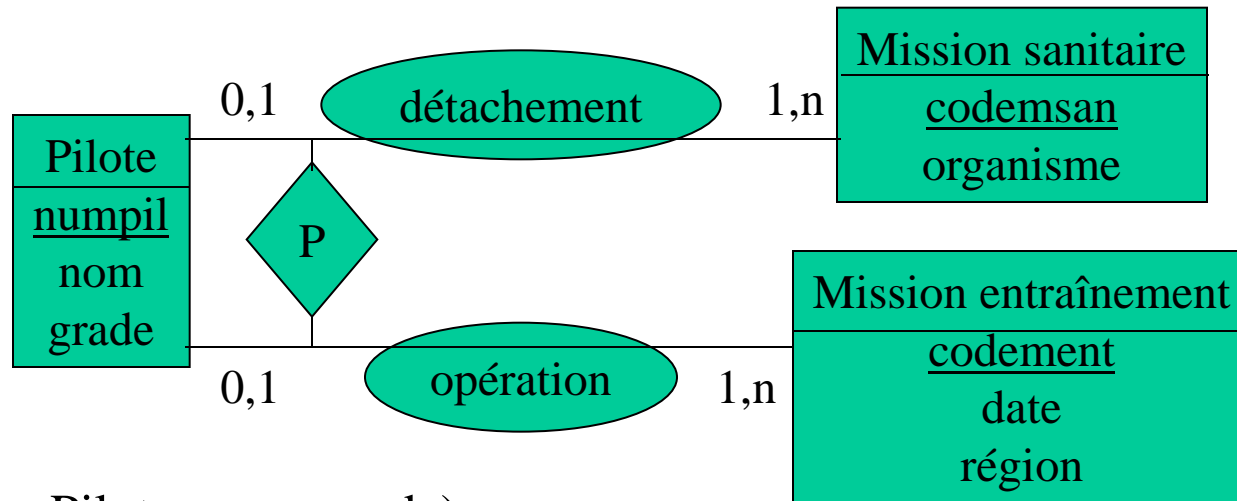




# ASSOCIATION N:M REFLEXIVE

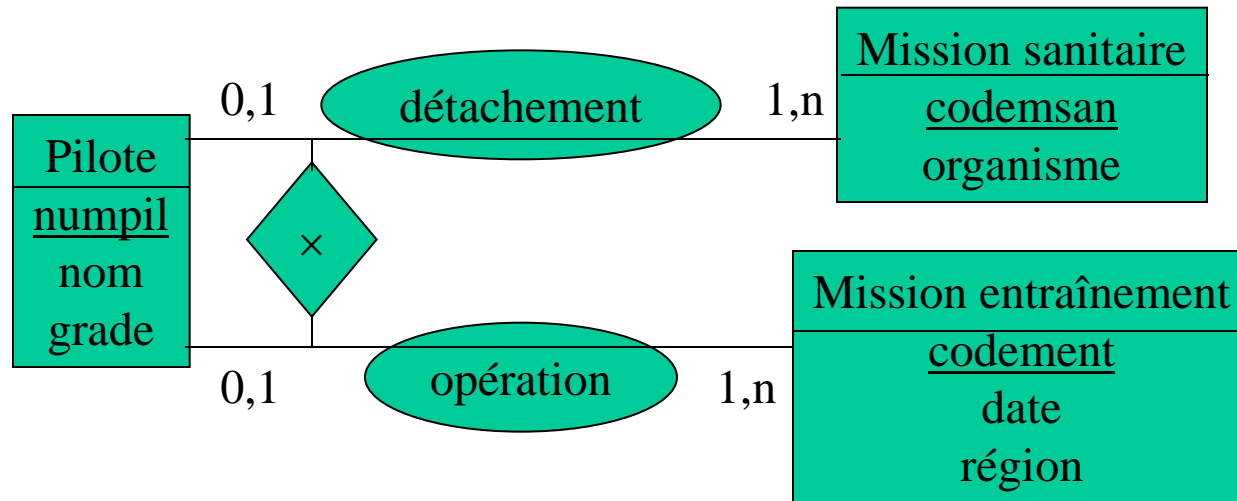


# Contrainte de partition



- Pilote (numPilote, nom, grade)
- MissionSanitaire(codemsan, organisme)
- MissionEntraînement (codement, date, région)
- Détachement (#numpilote, #codemsan)
- Opération (#numpilote, #codement)
- $\text{Pilote}[\text{numPilote}] \subseteq \text{Détachement}[\text{numPilote}] \cup \text{Opération}[\text{numPilote}]$
- $\text{Opération}[\text{numPilote}] \cap \text{Détachement}[\text{numPilote}] = \emptyset$

# Contrainte d'exclusion



☆Pilote (numPilote, nom, grade)

☆MissionSanitaire(codemsan, organisme)

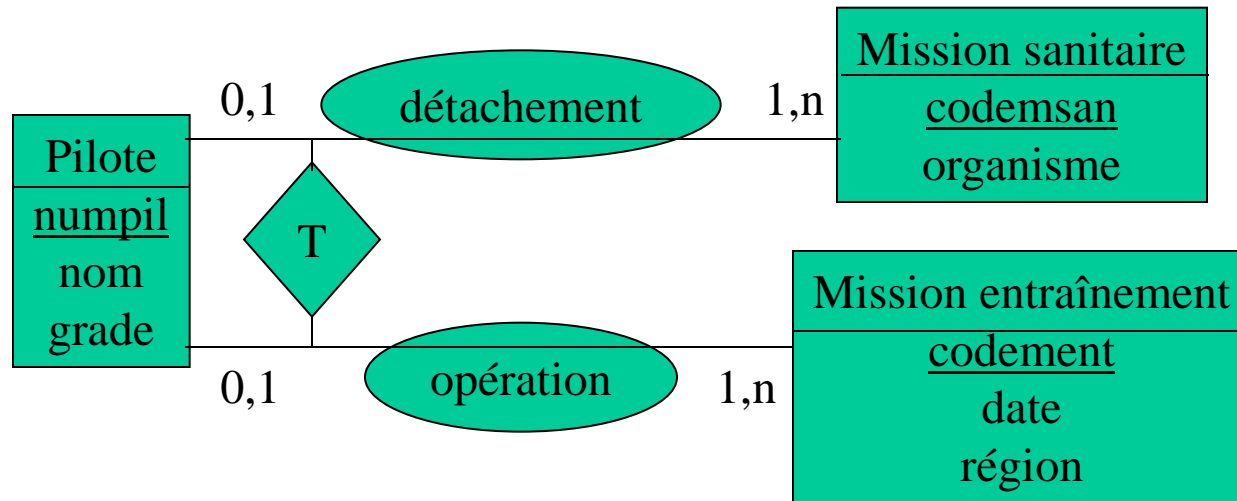
☆MissionEntraînement (codement, date, région)

☆Détachement (#numpilote, #codemsan)

☆Opération (#numpilote, #codement)

☆Opération[numPilote]  $\cap$  Détachement[numPilote] =  $\emptyset$

# Contrainte de totalité



☆Pilote (numPilote, nom, grade)

☆MissionSanitaire(codemsan, organisme)

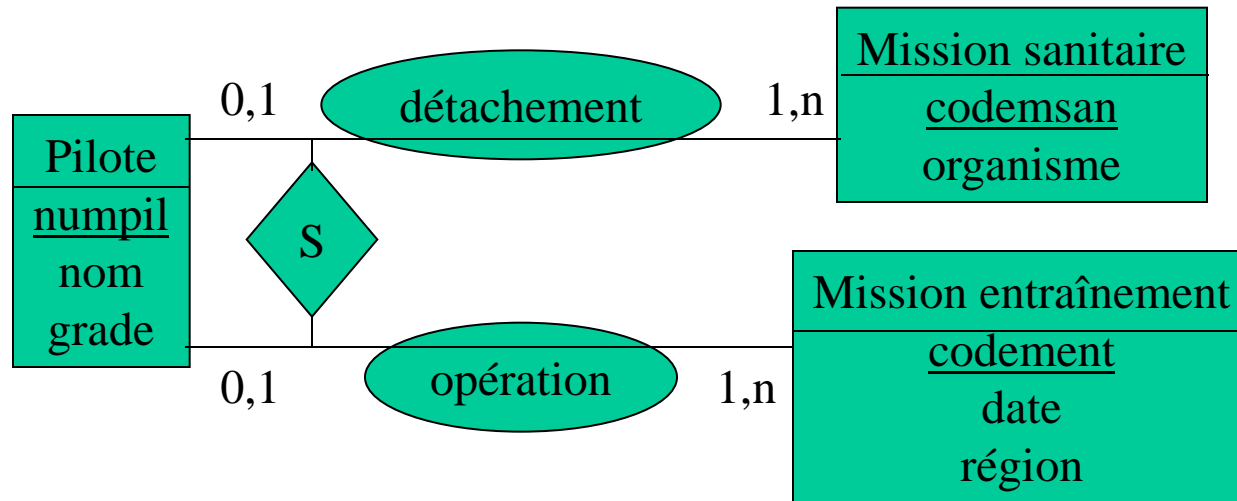
☆MissionEntraînement (codement, date, région)

☆Détachement (#numpilote, #codemsan)

☆Opération (#numpilote, #codement)

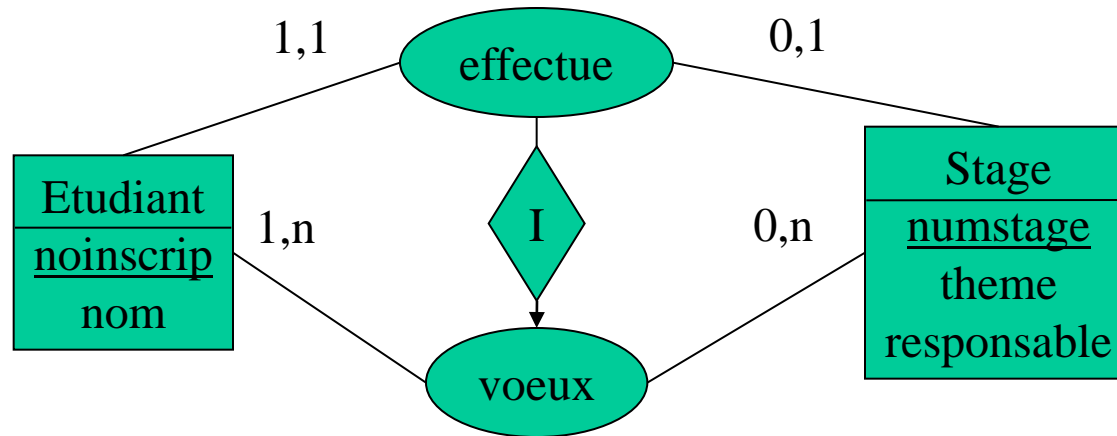
☆Pilote[numPilote]  $\subseteq$  Opération[numPilote]  $\cup$  Détachement[numPilote]

# Contrainte de simultanéité



- ☆Pilote (numPilote, nom, grade)
- ☆MissionSanitaire(codemsan, organisme)
- ☆MissionEntraînement (codement, date, région)
- ☆Détachement (#numpilote, #codemsan)
- ☆Opération (#numpilote, #codement)
- ☆Opération[numPilote] = Détachement[numPilote]

# Contrainte d'inclusion



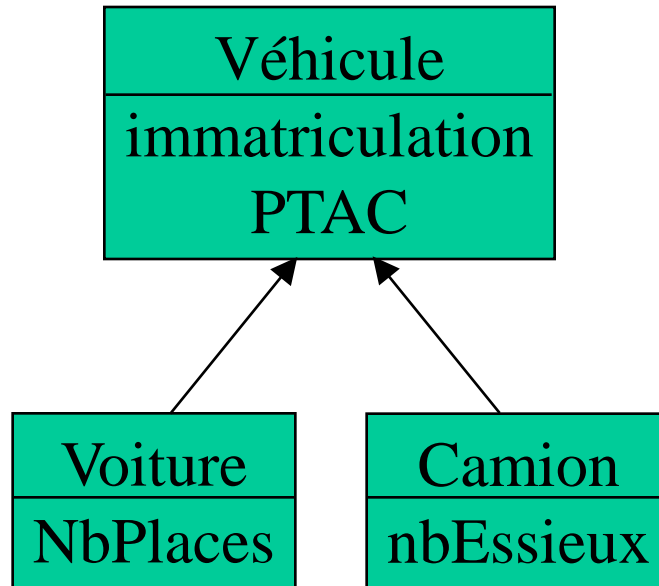
☆ Etudiant (noinscrip, nom, #numstage)

☆ Stage(numstage, theme, responsable)

☆ Voeux (noinscrip, numstage)

☆ Etudiant[noinscrip, numstage]  $\subseteq$  Voeux

# Héritage



- Véhicule (immatriculation , PTAC, type *obligatoire*)
- Voiture (#immatriculation, nbPlaces)
- Camion (#immatriculation, nbEssieux)

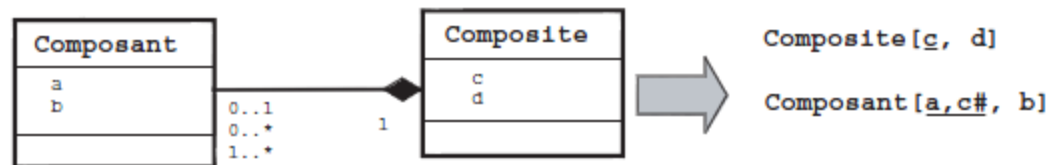
# Contraintes d'héritage

- Partition :
  - Véhicule[immatriculation]  $\subseteq$  Voiture[immatriculation]  $\cup$  Camion[Immatriculation]
  - Voiture[immatriculation]  $\cap$  Camion[Immatriculation] =  $\emptyset$
- Totalité :
  - Véhicule[immatriculation]  $\subseteq$  Voiture[immatriculation]  $\cup$  Camion[Immatriculation]
- Exclusion
  - Voiture[immatriculation]  $\cap$  Camion[Immatriculation] =  $\emptyset$



# Agrégation UML

- Alors que l'agrégation partagée de UML se traduit au niveau logique comme une simple association, il n'en est pas de même pour la composition.
- L'exemple décrit le schéma relationnel déduit d'une composition (on suppose que l'attribut a identifie la classe composante et que l'attribut c identifie la classe composite).
- La clé primaire des relations déduites des classes composantes doit contenir l'identifiant de la classe composite (quelles que soient les multiplicités).



# Agrégation & relations

L'agrégation est une forme qui permet d'abstraire la complexité d'un objet en cachant sa composition en d'autres objets. Elle permet de définir des structures d'objets complexes. Elle permet de voir les choses à différents niveaux de détail. (hiérarchie d'agrégation).

L'agrégation dans le modèle relationnel s'appuie sur la notion de clé.

Une clé est un attribut ou un groupe d'attribut qui identifie chaque nuplet de manière unique. Toute relation doit avoir une clé. Les clés servent à établir des liens entre relation.

L'agrégation telle que proposée dans le modèle relationnel consiste à attacher des attributs à une table. La clé d'une relation composite devient clé étrangère de la relation composée et partie de clé primaire.

**L'imbrication des structures ne peut alors se faire que par jointure de tables**

# Héritage & relations

Le modèle relationnel ne contient pas la forme d'abstraction

Généralisation/Spécialisation. Elle n'est pas remplaçable - sans perte de sémantique - par une forme agrégative quelle qu'elle soit.

**problème une entité générique et une entité spécifique sont 2 niveaux d'abstraction d'une même entité.**

On recommande de réfléchir dans chaque cas à une solution spécifique.

Tentatives de traductions : 1G, 2S

- Distinction : création de tables pour G&S avec les attributs spécifiques dans S, la clé primaire de G étant clé primaire et étrangère dans S
- Variante Distinction : création de tables G&S avec les attributs communs et spécifiques dans S
- Push-down : création uniquement de S en migrant tous les attributs de G
- Push-up : création uniquement d'une table avec des vues (amélioration) en migrant tous les attributs de(s) S dans G. Une amélioration en cas de disjonction est d'ajouter un attribut discriminant pour une meilleure abstraction des traitements

La facilité d'implantations des contraintes d'héritage peut être un critère de choix.

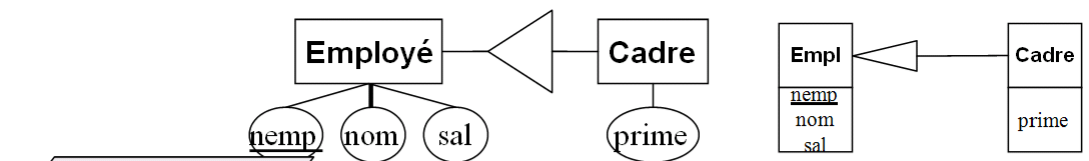
Lien G & S : même cycle de vie.

# Exemples de traduction de l'héritage vers le modèle relationnel

## Inconvénients

- 1 • Jointure pour reconstituer Cadre
  - 2 • Union pour reconstituer Employé
  - 3 • Pas de distinction entre un Employée et un Cadre avec une prime NULL
- en général, pas d'abstraction dans les Traitements

Option de traduction	Compatibilité avec les contraintes	
Distinction	OK	Exclusivité : {incomplete, disjoint} (par défaut)
	OK	Pas de contrainte : {incomplete, overlapping}
Push-down	OK	Partition : {complete, disjoint}
	KO	Totalité : {complete, overlapping}, redondance d'informations
Push-up	OK	Totalité : {complete, overlapping}
	KO	Autres cas : présence de nombreuses valeurs NULL



1 - Separation

Emp	<u>nemp</u>	nom	sal
	100	Dupond	5000
	200	Durant	20000

Cadre	<u>nemp</u>	prime
	200	10000

Cadre.nemp est une clé étrangère sur Emp

### 2 - Push down

Emp	<u>nemp</u>	nom	sal
	100	Dupond	5000

Cadre	<u>nemp</u>	nom	sal	prime
	200	Durant	20000	10000

CI : {Emp.nemp} ∩ {Cadre.emp} = ∅

### 3 - Push Up

Emp	<u>nemp</u>	nom	sal	prime
	100	Dupond	5000	NULL
	200	Durant	20000	10000

### 4

Emp	typemp	<u>nemp</u>	nom	sal	prime
	EMP	100	Dupond	5000	NULL
	CAD	200	Durant	20000	10000

Emp	<u>typemp</u>
	EMP
	CAD

Emp.typemp est une clé étrangère  
CI : Emp.typemp = EMP ⇒ prime = NULL

### 5

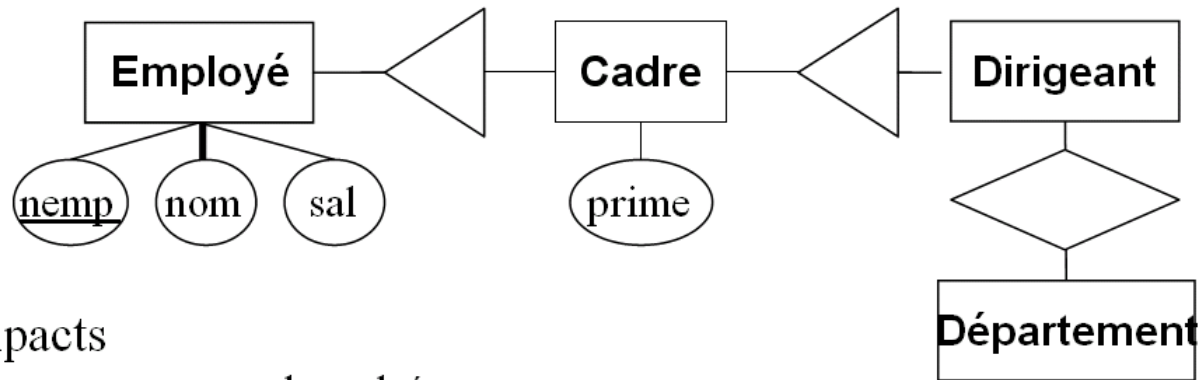
Emp	typemp	<u>nemp</u>	nom	sal	prime
	EMP	100	Dupond	5000	NULL
	CAD	200	Durant	20000	10000

Domaine typemp, CD typeemp IN (EMP, CAD)  
CI : Emp.typemp = EMP ⇒ prime = NULL

# Héritage et évolution de schéma

- Inconvénients

- Ajout d'une sous-entité Dirigeant

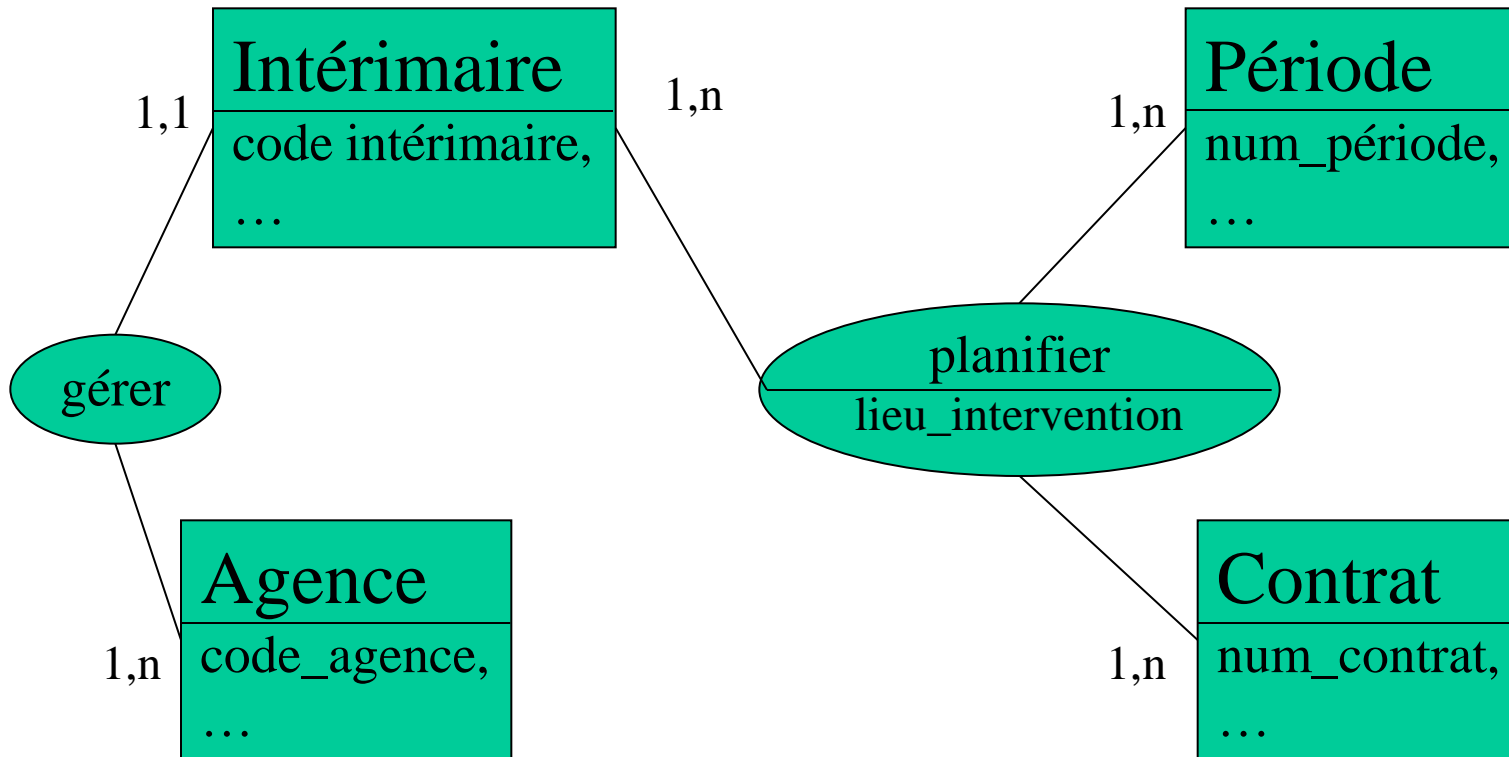


- Impacts

- sur le schéma
    - sur les contraintes d'intégrité
    - sur les contraintes de domaines
    - sur les traitements

# Transformation du MCD en MLD relationnel

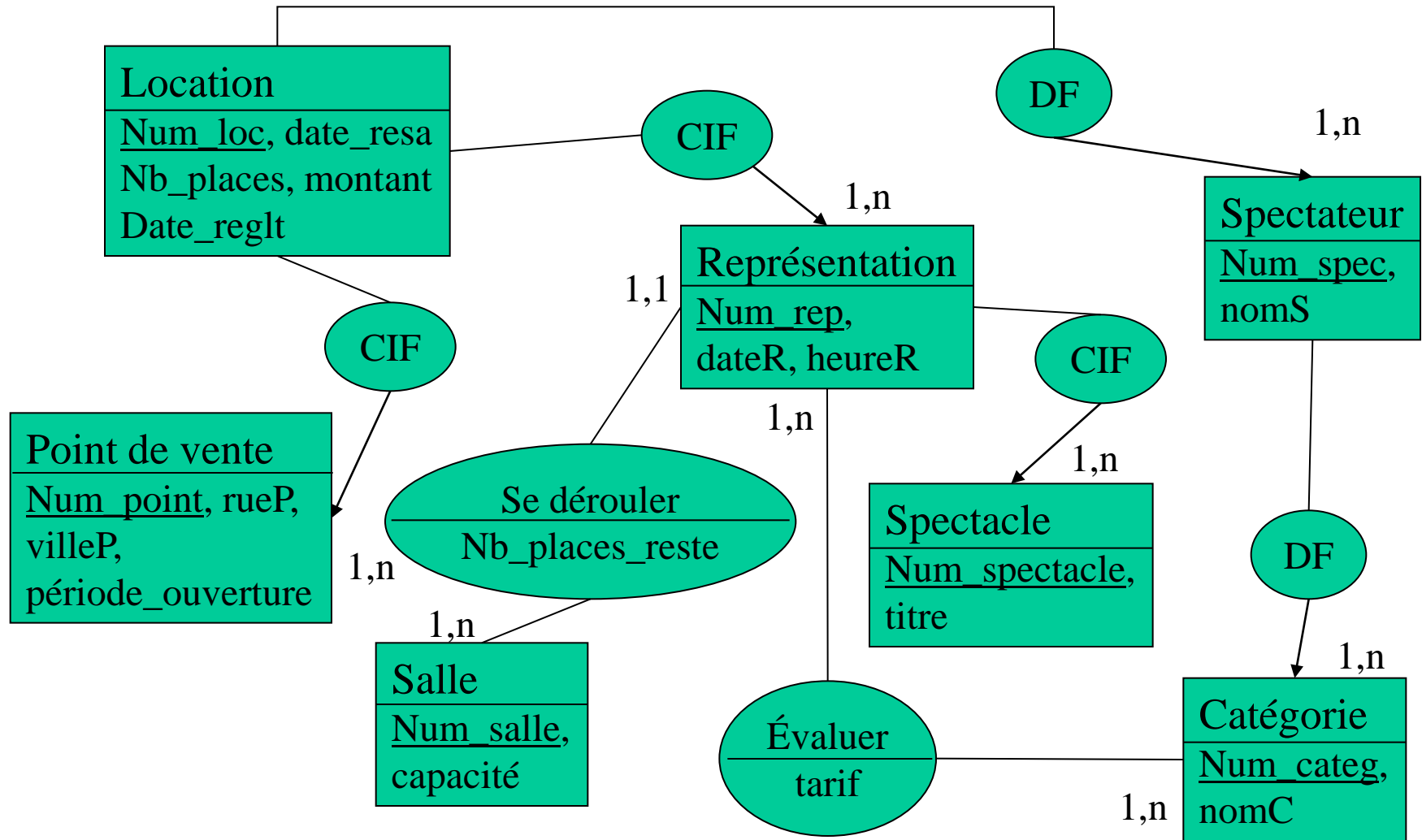
- Un exemple



# Transformation du MCD en MLD relationnel

- Le MLD relationnel qui en découle
- Entités
  - Agence(codeA, nomA, rueA, villeA)
  - Contrat(num\_contrat, caractéristiques)
  - Intérimaire(codeI, nomI, rueI, villeI, #codeA)
  - Période(num\_période, ...)
- Associations
  - Planning (#codeI, #num\_contrat, #num\_période, lieu\_intervention)
  - L'association gérer n'apparaît pas directement : la clé étrangère codeA dans intérimaire la remplace

# Transformation du MCD en MLD relationnel







# Chapitre 7

## **Autres formalismes pour AGL base de données Utilisation concrète, pratique**

Michel Dubois

I.U.T. de Vannes

*Michel.Dubois@univ-ubs.fr*

[Retour au plan](#)



# UML, temps et espace géographique (état courant)

- Besoin de types
  - Type abstrait de données (TAD)
  - Pour gérer le temps
  - Pour gérer les géométries des objets géographiques
- Modélisation et gestion de la géométrie (faible)
  - Pas de types spécifiques pour la géométrie
  - Pas d'opérateurs spécifiques (inclusion, ...)
- Gestion du temps (moyenne)
  - Type abstrait (timestamp, dates ... possibles) en fonction de l'environnement
  - Opérateur  $< > = \text{plus}$

# TAD - Types Abstraits des Données

- Une vue fonctionnelle abstraite des objets spatiaux qui ne nécessitent pas la connaissance de la structuration interne de ces objets.
- L'interface externe du TAD est une liste d'opérations : un objet de type zone est vu comme la liste des opérations que l'on peut faire sur ces zones.
- Le langage de requête qui utilise le TAD spatial ne connaît pas les détails d'implémentation du TAD, c'est-à-dire ni la structure de représentation des objets spatiaux, ni la façon dont sont implémentées les opérations (adjacence, intersection, etc. ).
- Dans un modèle relationnel étendu aux TAD (qui connaît les opérations sur TAD), la requête suivante permet d'avoir les communes limitrophes de Vannes dans le même département :

```
- SELECT c1.nomCommune
   FROM Commune c1, Commune c2
   WHERE c1.nomCommune = 'Vannes'
   AND c1.numDepartement=c2.numDepartement
   AND Adjacent (c1.geometrie, c2.geometrie)
```

Michel Dubois

Type zone

...

Adjacent : zone x zone -> bool  
(teste l'adjacence de deux zones)

...

Type ligne...

Type point...

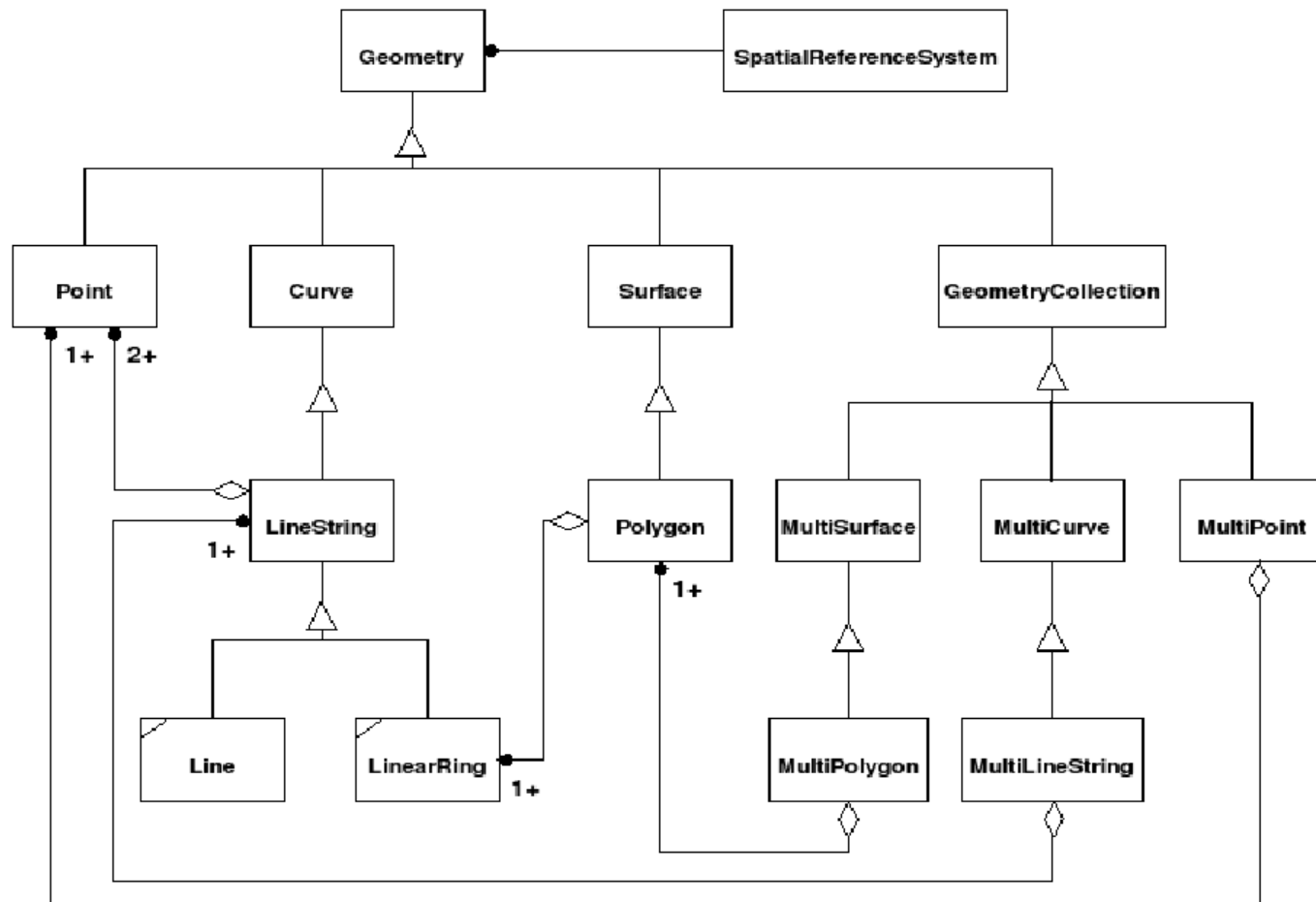
# Compléments objet des types abstraits de données (TAD)

- Collections
- Identité des objets par l'utilisation de références
- Héritage
- Encapsulation par la programmation de méthodes (procédures ou fonctions)

# SIG & SQL 2003

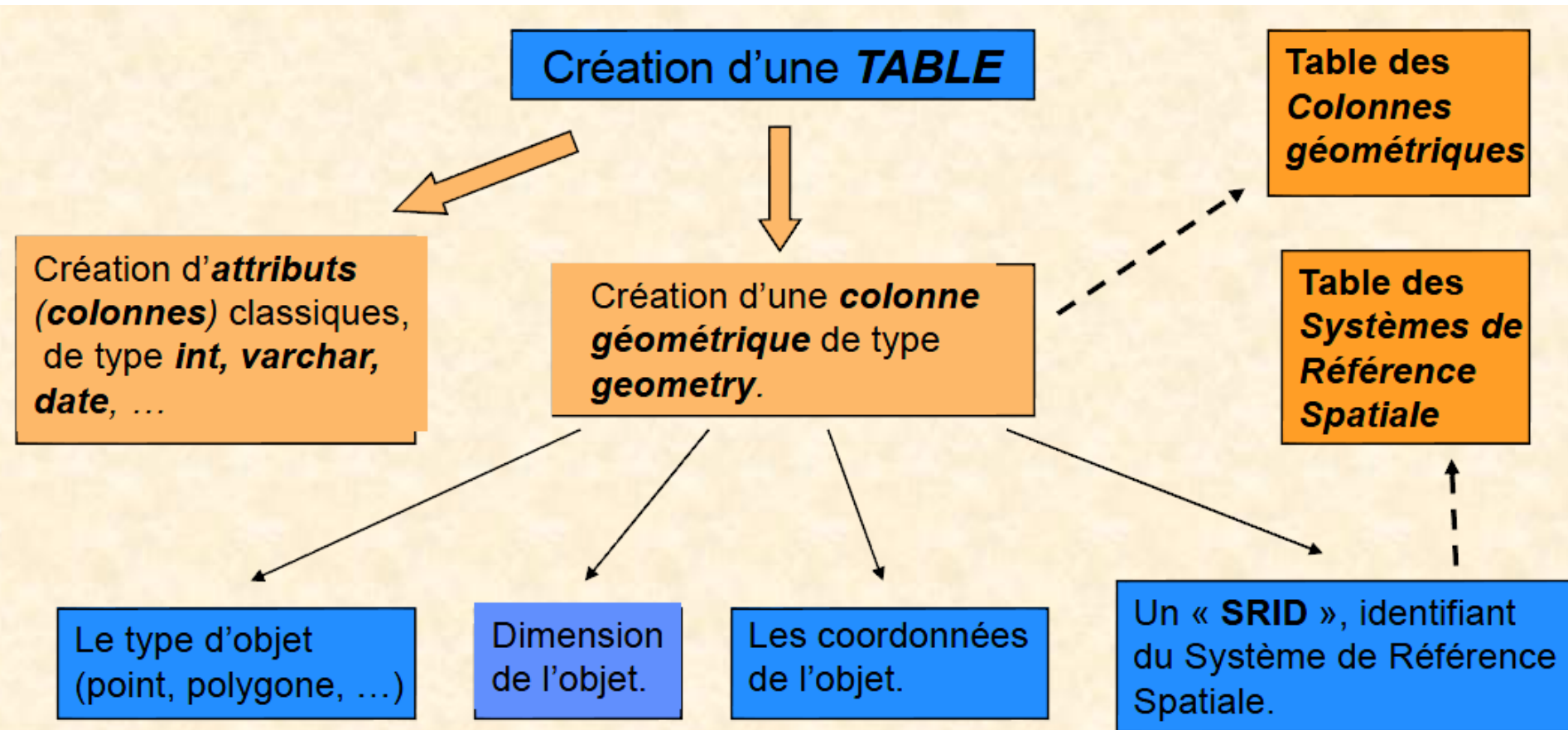
- Le « Structured Query Language » SQL ne possède pas à l'origine d'opérateurs spatiaux. Les constructeurs ont étendu ce langage afin de permettre la mise en oeuvre de requêtes spatiales.
- Bien qu'il existe maintenant des normes, il y a une palette d'appellations différentes entre les logiciels.
- Normes issues des SGBD – SQL MM  
ISO/IEC 13249-3:2003, Information technology — Database languages — SQL multimedia and application packages — Part 3: Spatial. Le standard SQL/MM utilise le préfixe ST\_ pour toutes les tables, vues, méthodes et fonctions (ST Spatial Temporal to ST Spatial Type)
- Normes issues des SIG : Groupe TC211 – ISO 19xxx  
ISO 19125-1:2004, Geographic information — Simple feature access — Part 1: Common architecture
- Ces normes sont désormais coordonnées avec les standards de l' Open Geospatial Consortium (OGC)
- OpenGIS® Implementation Specification for Geographic information – Simple feature access
  - Simple Features - SQL - Binary Geometry
  - Simple Features - SQL - Normalized Geometry
  - Simple Features - SQL - Types and Functions
- Elles sont de plus en plus intégrées dans les SGBD spatiaux

# Modèle objet OGC / SQL MM



# Les principes objet-relationnels de gestion de l'information spatiale

PostgreSQL - PostGIS, Oracle, IBM DB2, MySQL (relationnel), MS SQL Server 2008



# Rappels requête relationnelle-objet

- Les extensions spatiales des SGBD Oracle, PostgreSQL sont basée sur la création d'un type utilisateur spécifique géométrique.
- L'utilisateur doit savoir manipuler des requêtes relationnelles-objet
- L'accès aux objets se fait obligatoirement en préfixant le chemin d'accès par l'alias de la table et non directement par le nom de la table.

```
SELECT alias_table.nom_objet.nom_attribut  
FROM nom_table AS alias_table;
```

```
SELECT alias_table.nom_objet.nom_méthode(paramètres)  
FROM nom_table AS alias_table;
```

```
UPDATE nom_table AS alias_table  
SET attribut = valeur  
... alias_table.nom_objet.nom_attribut = valeur  
... alias_table.nom_objet= objet  
WHERE attribut = valeur  
AND alias_table.nom_objet.nom_attribut = valeur  
AND alias_table.nom_objet.nom_méthode(paramètres)=retour  
;
```

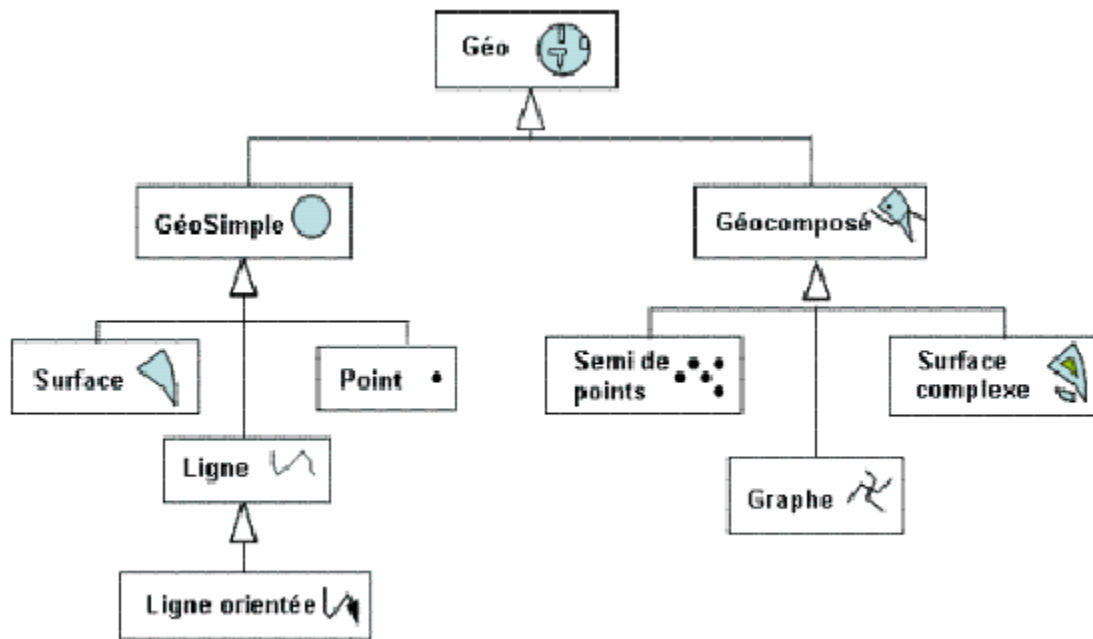




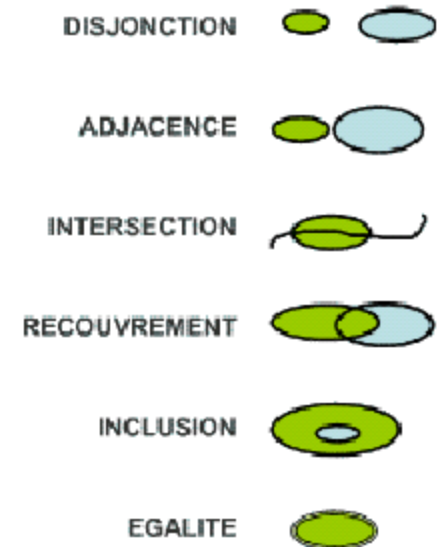
# Extension à la modélisation objets temporels et géographiques

- MADS (Modélisation d'Applications à Données Spatiales : formalisation plus aboutie)
  - EPFL (Parent, Christine ; Spaccapietra, Stefano)
  - Intégration à Oracle, Arcview et MapInfo
- Perceptory (« dépôt des perceptions de l'utilisateur », environnement logiciel plus abouti et plus proche d'UML)
  - Repose sur un langage de pictogrammes PictograF (gestion des caractéristiques spatiales et temporelles des classes d'objets, attributs et relations)
  - Université Laval, Québec, CA (Bédard, Yvan)
  - Sous forme de gabarit supporté par le logiciel Visio (Microsoft) pour générer des modèles de données Oracle 9i et Oracle 10g
  - PictograF aussi intégrable à ArgoUML (UML 1.4) et à Visual Paradigm
- UML-GeoFrame (logiciel ArgoCASEGEO)
  - Federal University of Viçosa, Brazil (Jugurta, Lisboa, Filho)

# MADS (Modélisation d'Applications à Données Spatiales )



*Hiérarchie des types abstraits de données spatiaux du modèle MADS*

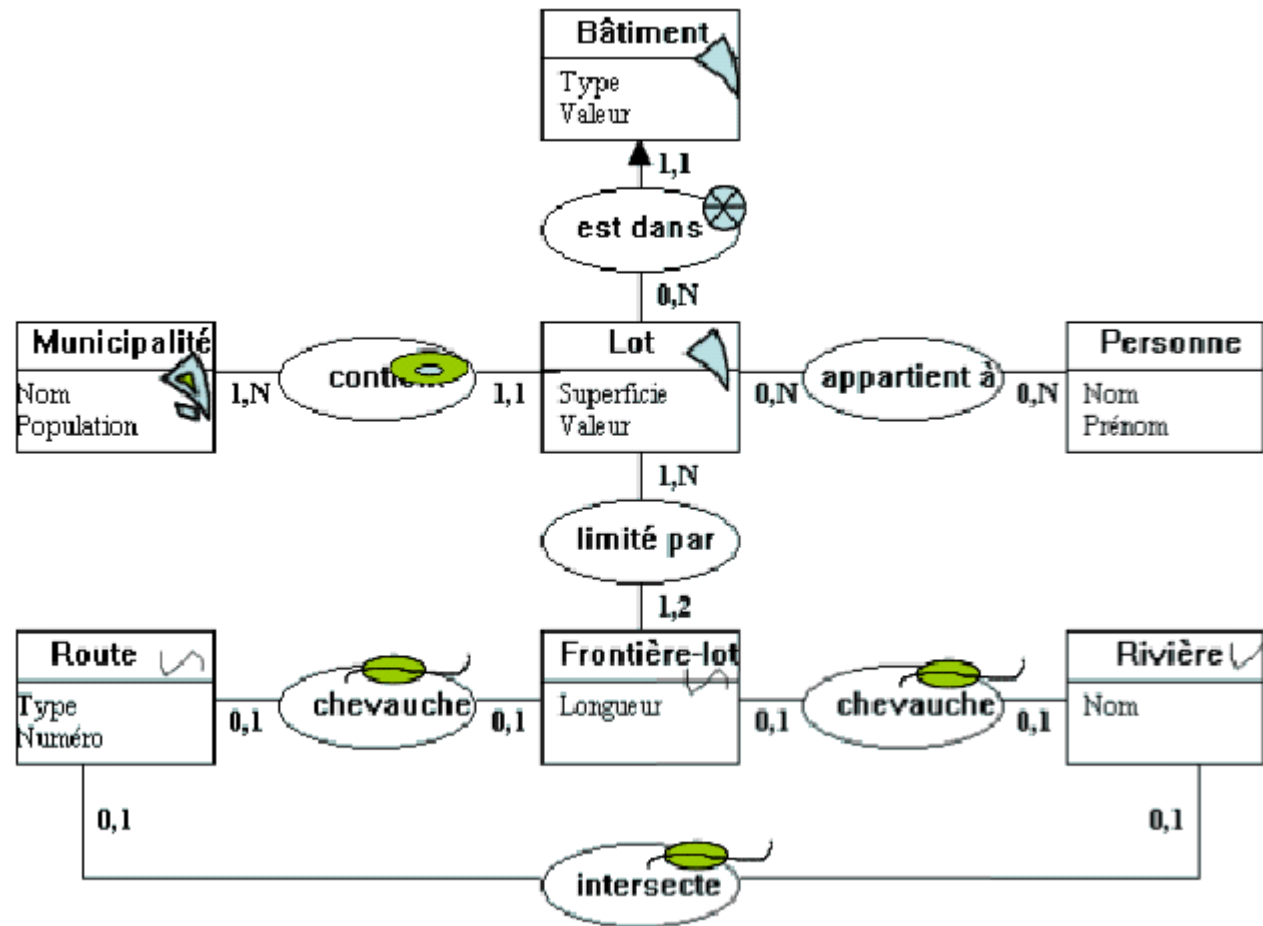


*Pictogrammes des relations topologiques (modèle MADS)*



# Exemple des municipalités du canada

## MADS



Exemple de schéma MADS

# PictograF

DIMENSIONNÉLITÉ DE LA FORME	DIMENSIONNÉLITÉ DE L'UNIVERS			
	NON EXPLICITE	1D	2D	3D
NON EXPLICITE				
0D (FORME PONCTUELLE)				
1D (FORME LINÉAIRE)				
2D (FORME POLYÉDRALE)				
3D (FORME VOLUMIQUE)				

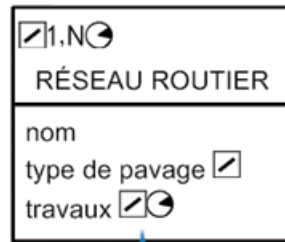
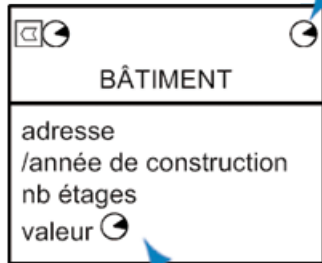
Tableau 1: Les pictogrammes spatiaux

DIMENSIONNÉLITÉ TEMPORELLE	PERSISTENCE		
	NON EXPLICITE	NON VOLATILE	VOLATILE
NON EXPLICITE			
8D INSTANT			
1D DURÉE			

Tableau 2: Les pictogrammes temporels

Évolution spatiale de la classe

Existence de la classe

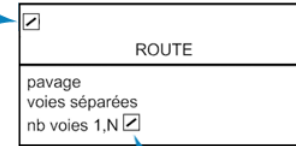


Évolution descriptive de l'attribut

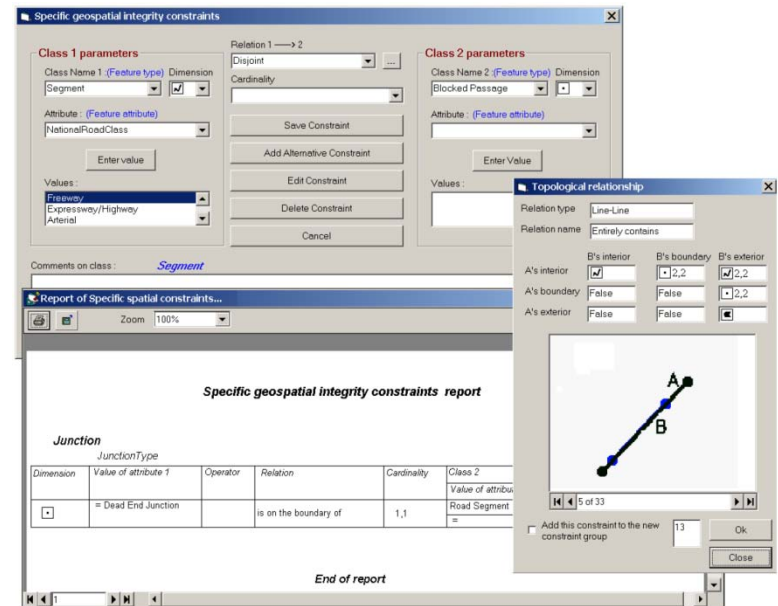
Évolution spatiale de l'attribut

- PictograF permet aussi d'exprimer des contraintes spatiales

Géométrie définie pour la classe d'objets

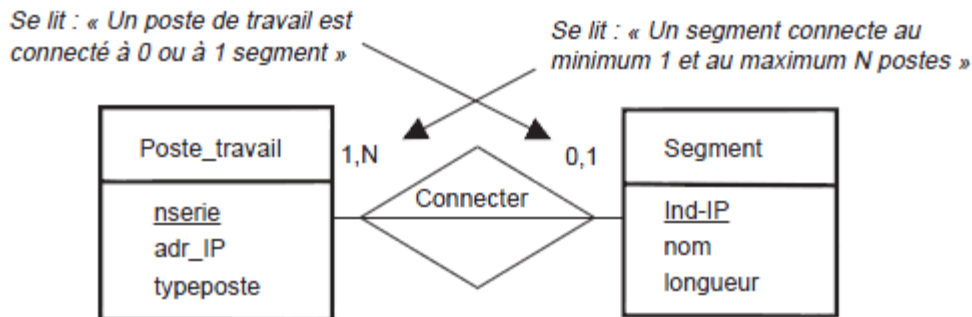


Géométrie définie pour un attribut

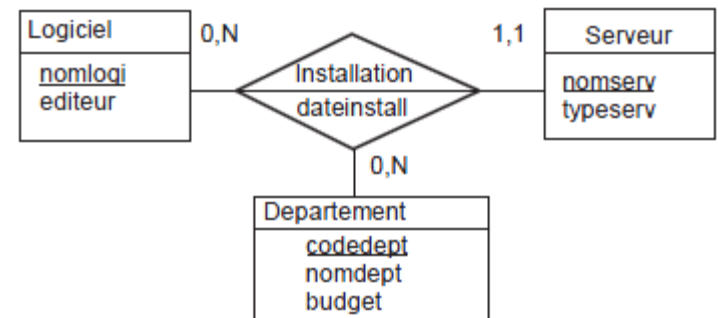


# Formalisme de P. CHEN

- L'interprétation des cardinalités constitue la différence fondamentale entre le formalisme du modèle de P. Chen et le MCD de Merise.
- Les cardinalités d'une association binaire dans le modèle de Chen et de Merise sont inversées au niveau de l'axe de représentation de l'association. Les cardinalités d'une association binaire dans le modèle de Chen et dans le formalisme UML sont positionnées de façon identique.
- Le problème de l'approche de P. Chen réside dans son manque de cohérence entre la représentation des associations binaires et des associations *n-aires*. *La majorité des outils de conception américains n'ont pas suivi cette vision des choses car ils ont été incapables de programmer ce concept (même le produit Designer d'Oracle). Ces outils modélisent les associations n-aires en définissant  $n+1$  entité(s), dont  $n$  sont reliées à une seule par des associations*



Michel Dubois

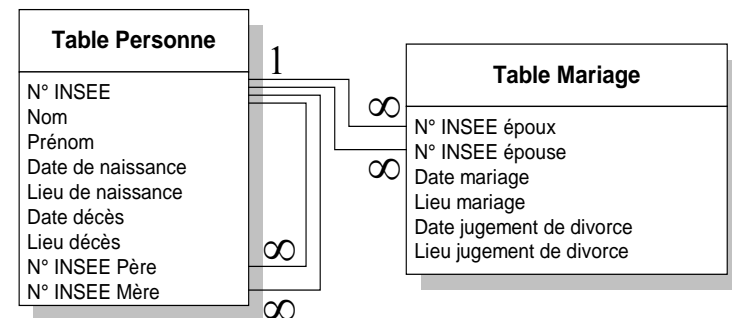
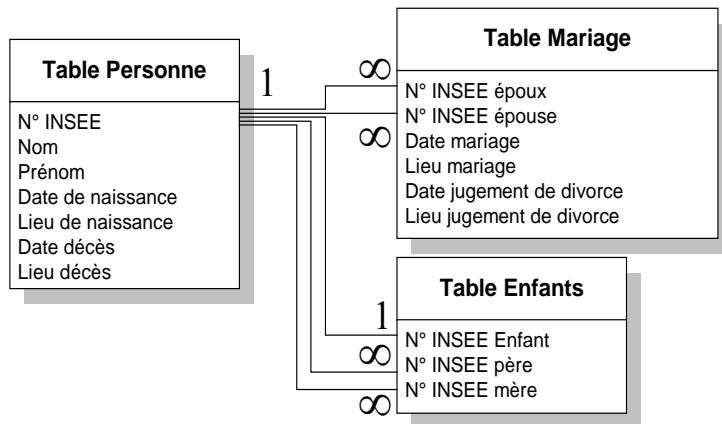


# *TYPES DE CONTRAINTES D'INTÉGRITÉS RÉFÉRENTIELLES*

- Contraintes 1 à plusieurs ( $\infty$ ) : La clé primaire (côté 1) ne peut contenir de doublons. La clé étrangère (côté  $\infty$ ) peut quand à elle contenir des valeurs en double pour peu qu'elles existent dans la clé primaire associée.
- Contraintes 1 à 1 : Ces contraintes sont plus rares et correspondent au cas où la clé étrangère ne peut contenir de doublons.

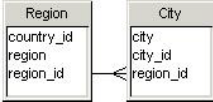

# *TYPES DE CONTRAINTES D'INTÉGRITÉS RÉFÉRENTIELLES*

- Représentations d'un schéma de tables avec des contraintes d'intégrité référentielles dans le cadre d'une base de données mémorisant l'Etat Civil.



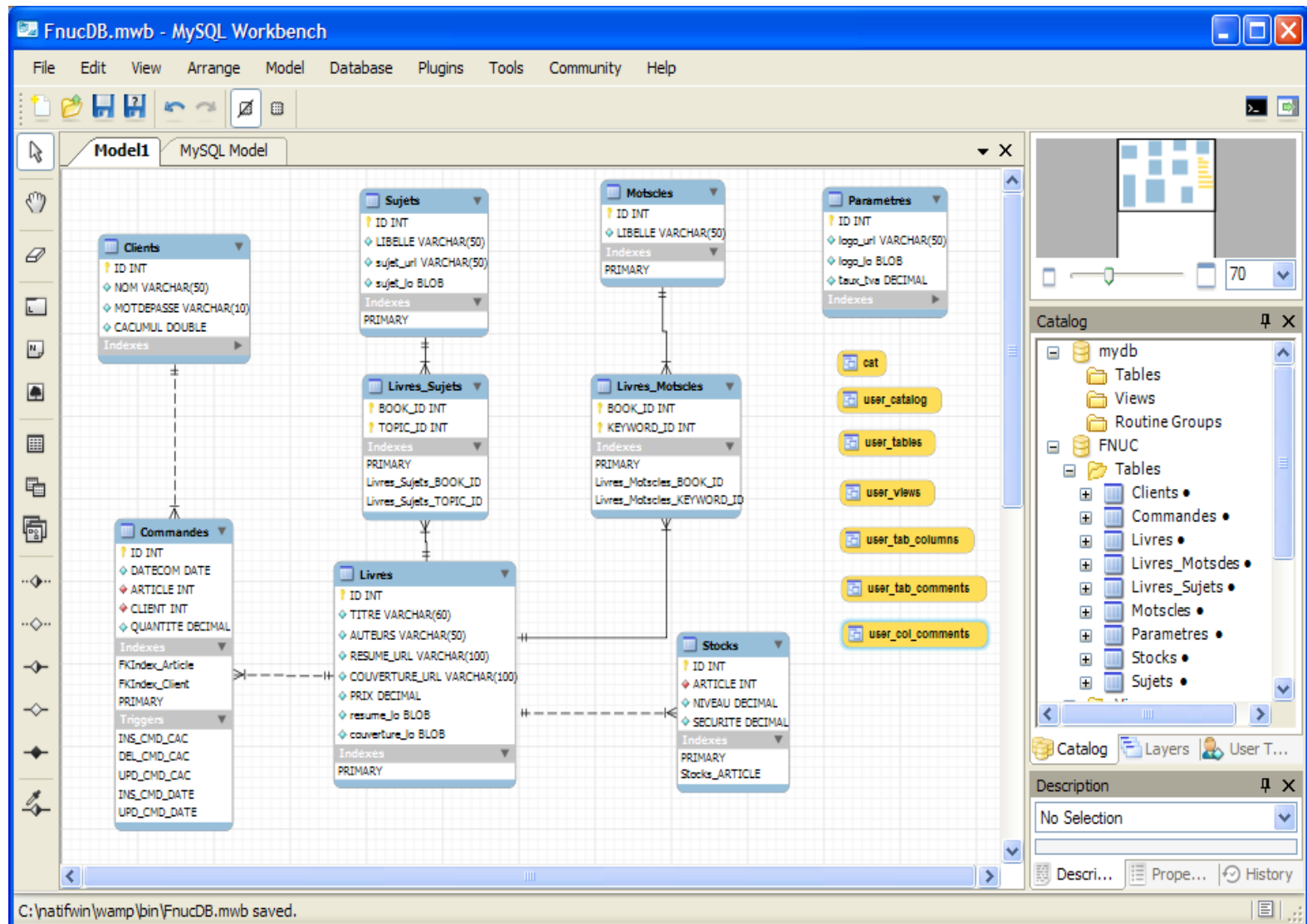
# AUTRES NOTATIONS DE TYPES DE CONTRAINTES D'INTÉGRITÉS RÉFÉRENTIELLES

La notation précédente est présente dans le logiciel MS ACCESS. Il existe d'autres notations qui sont présentes dans les logiciels de conception des bases de données (DB DESIGNER 4) ou dans Business Object:

Notation	Exemple	Description
Arité Information Engineering (Martin ERD)		Une patte d'oie indique l'extrémité "plusieurs" de la jointure. Elle correspond à $\infty$ dans Microsoft Access. L'extrémité « plusieurs » correspond à la présence de doublons au niveau de la clé étrangère. Si la cardinalité (max,max) est (1,1), une ligne droite s'affiche. La clé étrangère n'accepte pas de doublon.
Cardinalite  MERISE 1,N		La cardinalité est représentée sous la forme d'un ratio à chaque extrémité de la jointure. Le n se trouve sur l'autre extrémité que le $\infty$ .
Multiplicité UML, OMT 1..*		Les multiplicités sont inversés par rapport à MERISE.



# MySQL WORKBENCH



# Information Engineering (IE par Martin)

Le modèle est binaire : les associations n-aires ne sont pas autorisés et les associations sont non porteuses d'attributs. Dans le cas contraire, elles deviennent des entités associatives (entités avec un losange).

Les attributs ne sont pas représentés.

Le ou-exclusif entre deux association est une association avec un cercle noir médian qui sépare les entités exclusives

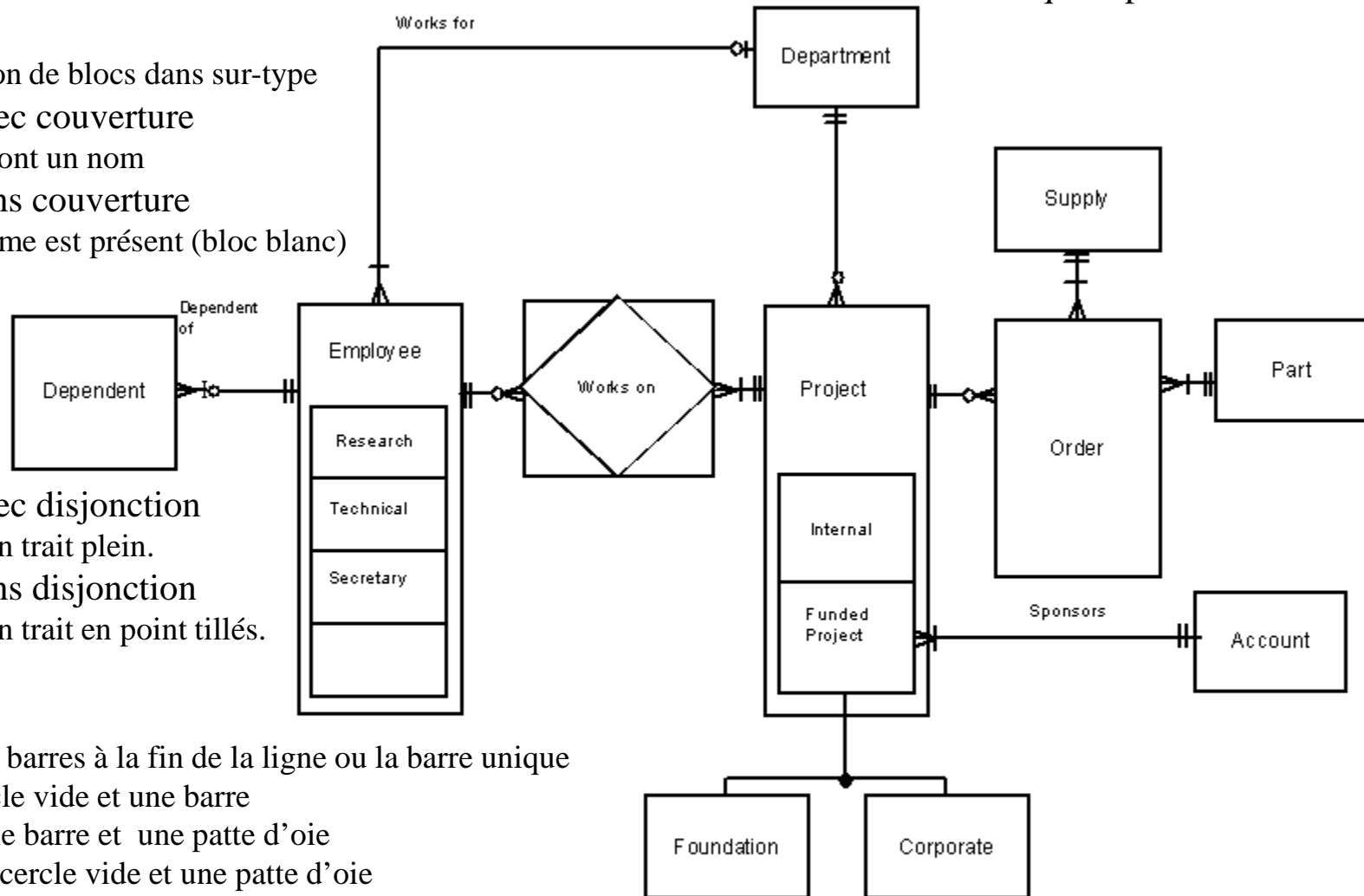
Héritage : inclusion de blocs dans sur-type

Classification avec couverture

tous les sous-types ont un nom

Classification sans couverture

un sous-type anonyme est présent (bloc blanc)



Cardinalités :

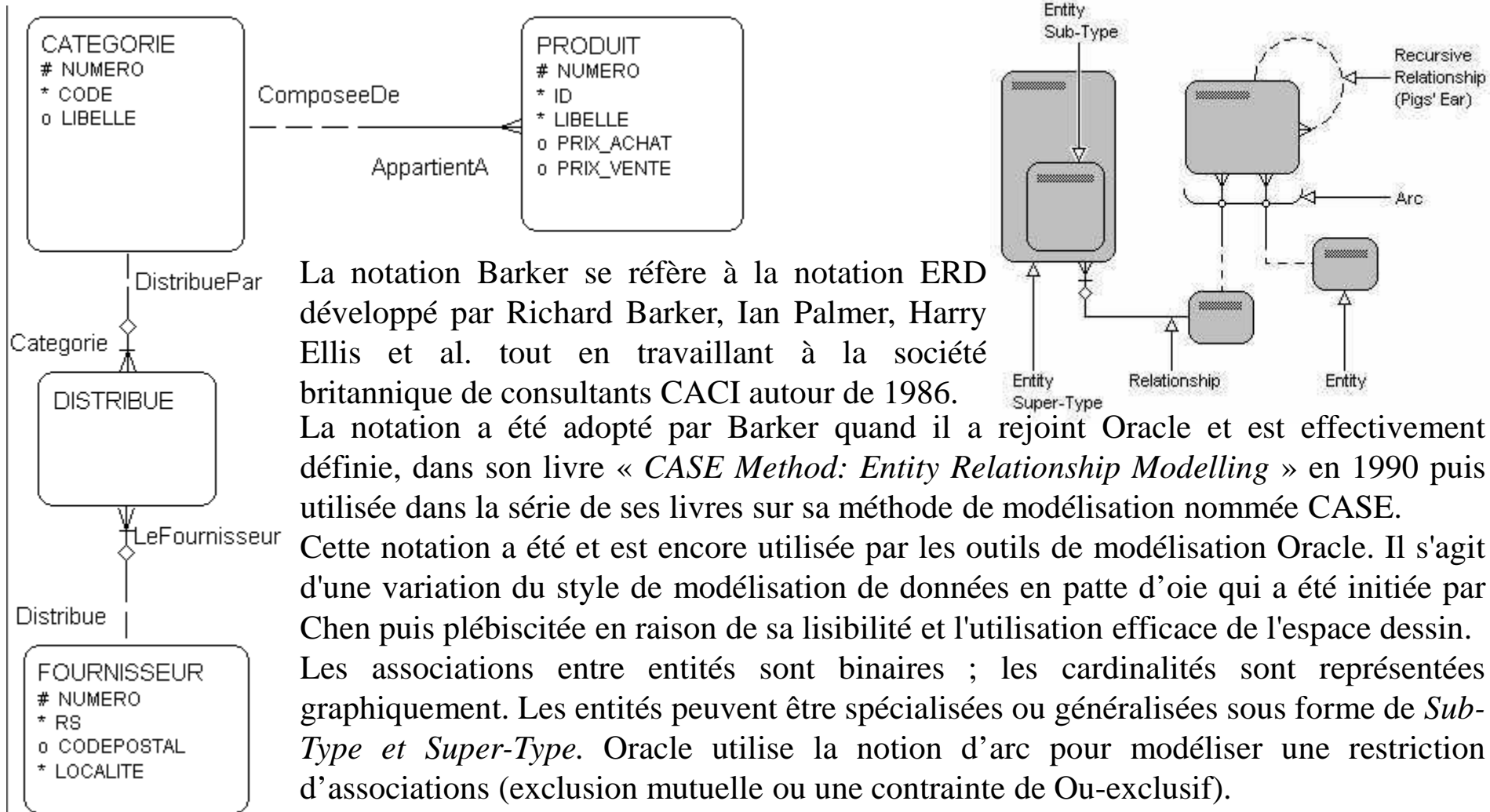
un et un seul : deux barres à la fin de la ligne ou la barre unique

zéro ou un : un cercle vide et une barre

un ou plusieurs : une barre et une patte d'oie

zéro, plus d'un : un cercle vide et une patte d'oie

# Barker's Notation



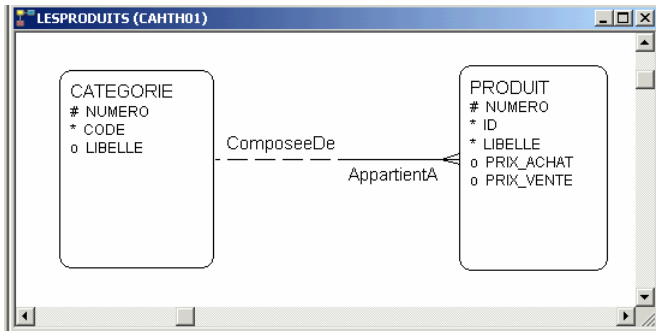
La notation Barker se réfère à la notation ERD développé par Richard Barker, Ian Palmer, Harry Ellis et al. tout en travaillant à la société britannique de consultants CACI autour de 1986.

La notation a été adoptée par Barker quand il a rejoint Oracle et est effectivement définie, dans son livre « *CASE Method: Entity Relationship Modelling* » en 1990 puis utilisée dans la série de ses livres sur sa méthode de modélisation nommée CASE.

Cette notation a été et est encore utilisée par les outils de modélisation Oracle. Il s'agit d'une variation du style de modélisation de données en patte d'oie qui a été initiée par Chen puis plébiscitée en raison de sa lisibilité et l'utilisation efficace de l'espace dessin. Les associations entre entités sont binaires ; les cardinalités sont représentées graphiquement. Les entités peuvent être spécialisées ou généralisées sous forme de *Sub-Type* et *Super-Type*. Oracle utilise la notion d'arc pour modéliser une restriction d'associations (exclusion mutuelle ou une contrainte de Ou-exclusif).

Le modèle étant binaire (pas de n-aire), pour remplacer les associations de degré **n:n**, il y a des entités associatives ; les entités associatives sont des entités dépendantes, identifiées par deux parents ou plus. Une entité associative peut être dotée d'attributs et être, à son tour, associée à toute autre entité.

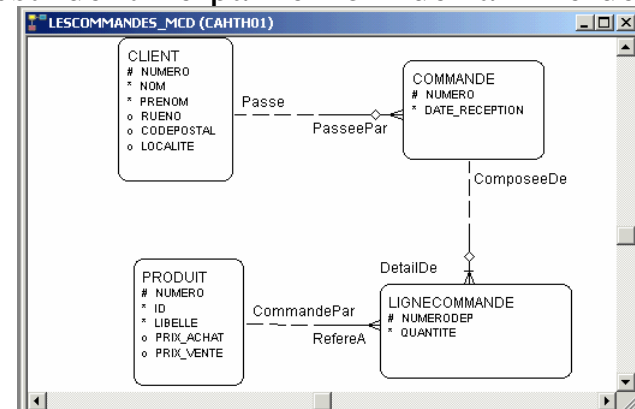
# Barker's Notation



- les rôles joués par les 2 entités participant à l'association ;
- la cardinalité minimale de 0, représentée par un trait tillé, pour le rôle **ComposeeDe** de l'entité **CATEGORIE** ;
- la cardinalité maximale de n, représentée par une patte d'oie sur le rôle opposé, pour le rôle **ComposeeDe** de l'entité **CATEGORIE** ;
- la cardinalité minimale de 1, représentée par un trait plein, pour le rôle **AppartientA** de l'entité **PRODUIT** ;
- la cardinalité maximale de 1, représentée par une absence de patte d'oie sur le rôle opposé, pour le rôle **AppartientA** de l'entité **PRODUIT**.

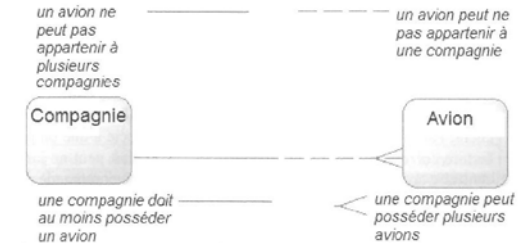
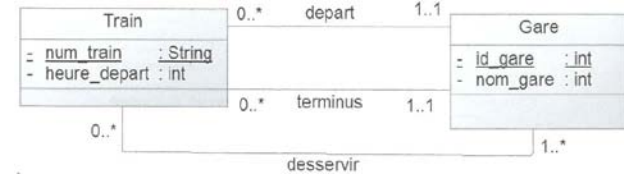
Lorsqu'une association est identifiante, elle est non transférable pour respecter la règle de stabilité de clé primaire.

- la non transférabilité de l'association entre **CLIENT** et **COMMANDE**, représentée par un losange; cette non transférabilité spécifie que lorsqu'une commande est créée et attribuée à un client, elle ne peut plus être attribuée ou transférée à un autre client. Ceci implique que si un utilisateur se trompe de client lors de l'établissement d'une commande, il devra détruire la commande erronée et en recréer une nouvelle en lui attribuant le client correct.
- l'association identifiante entre **COMMANDE** et **LIGNECOMMANDE**, représentée par un trait perpendiculaire à l'association ; l'association identifiante spécifie que les lignes de commandes seront identifiées par l'identifiant de la commande et par un identifiant de ligne tout comme un enfant est identifié par le nom de famille des parents et son prénom.



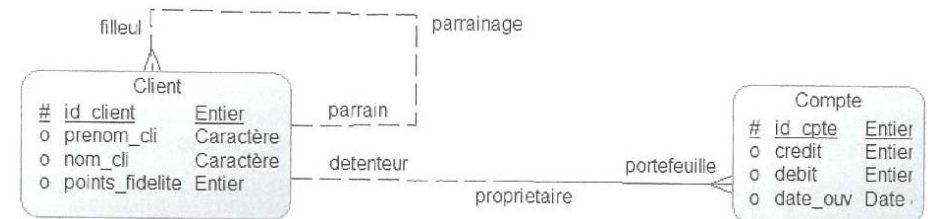
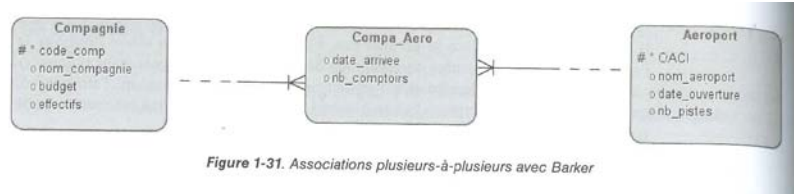
# Notation de Barker

Pour connaître la position des multiplicités d'une association binaire dans la notation de Barker, vous devez utiliser les deux côtés des liens simultanément.



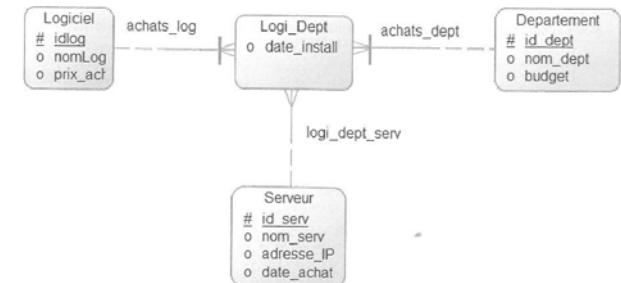
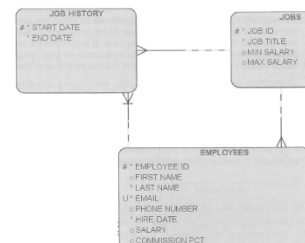
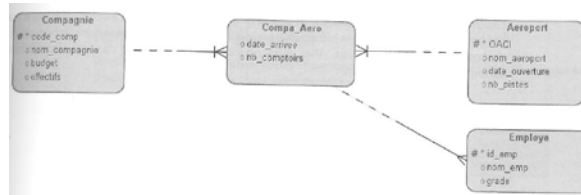
La notation de Barker permet également de positionner des rôles à l'extrémité des liens d'associations, Ils sont utiles lorsqu'il existe plusieurs occurrences participantes ou plusieurs associations entre les mêmes classes ou quand vous choisissez d'utiliser le langage OCL,

Les associations binaires porteuses d'attribut se traduit par une association d'identification afin que l'identifiant des couples sans doublons (compagnie,aéroport) est composé par les identifiant des entités concernées.



Une classe association liée (cas simple ou hiérarchie de classe-association) est facilement reliée à la troisième entité par une association classique (non identifiante)

Une association d'identification exprime facilement l'identification relative,



# Notation de Barker

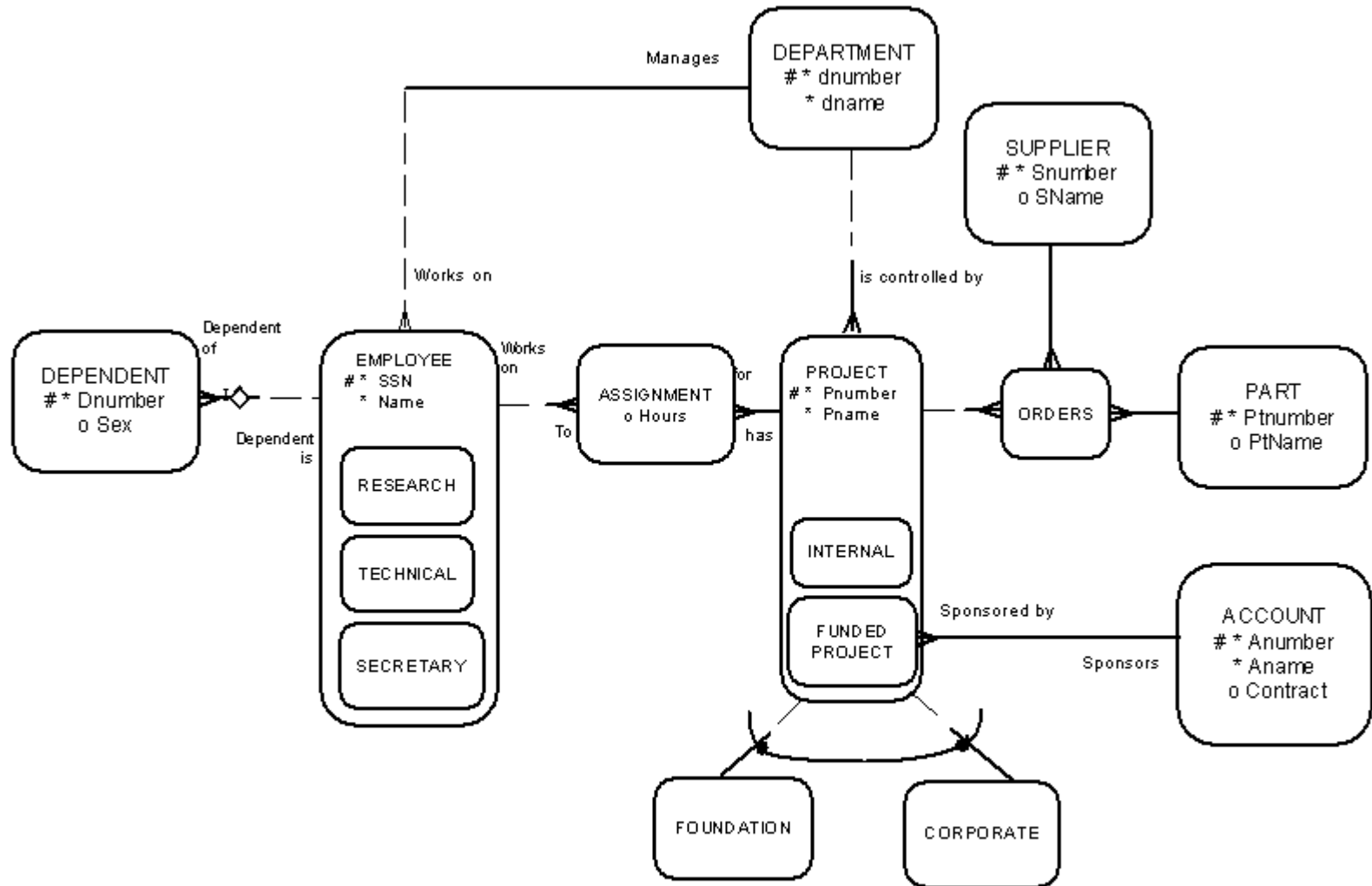


FIGURE 10: ORACLE's CASE\* METHOD Notation

# Notation de Bachman

- La méthode de Bachman modélise au niveau conceptuel les clés étrangères comme pour IDEF1X. Elle n'admet pas les association ternaire comme IDEF1X et Barker.
- Les rôles des attributs sont annotés en face de leur nom comme ceci :  
PK clé primaire, FK clé étrangère  
PFK clé primaire et étrangère, I hérité  
Cardinalités :  
max n = une flèche max 1 = une ligne  
min 0 = un cercle ouvert  
min 1 = un cercle rempli ou noir  
Quand une association n:m est non porteuse d'attribut, elle est représentée par une ligne. Sinon elle est modélisé comme un type d'entité.

Michel Dubois

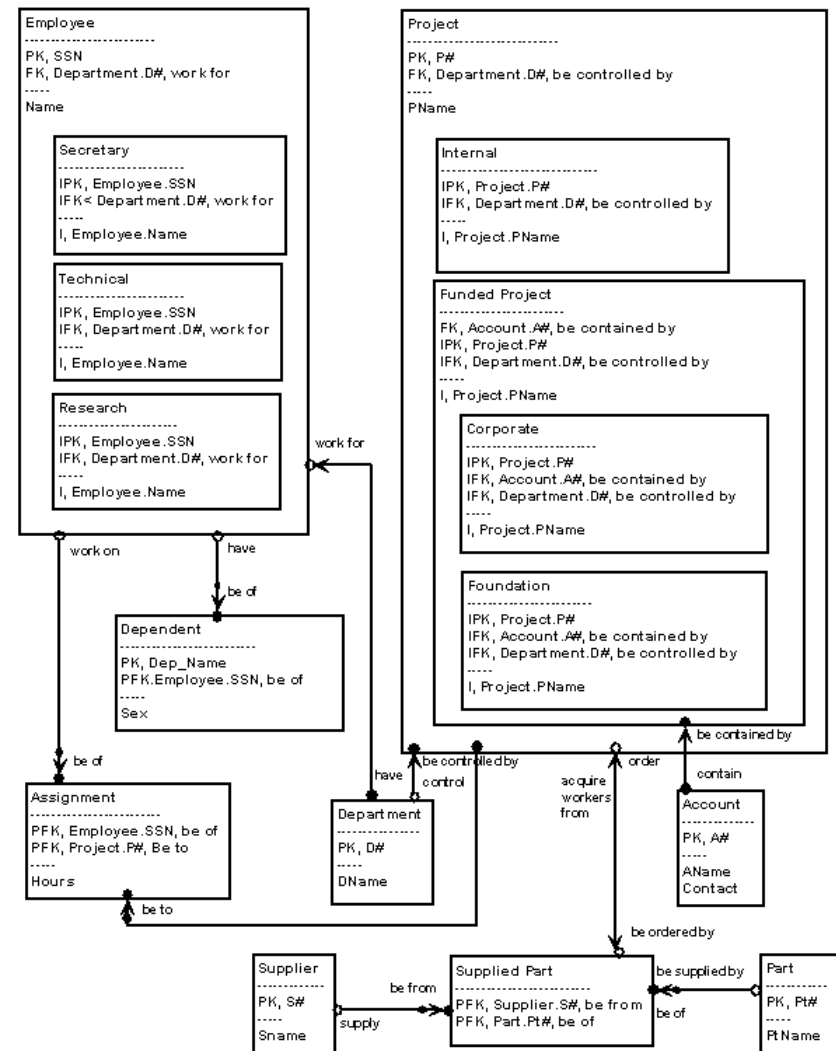


FIGURE 13: BACHMAN Notation

# Notation IDEF1X Information Model

- La méthode de IDEF1X modélise au niveau conceptuel les clés étrangères comme pour Bachman. Elle n'admet pas les association ternaire comme Bachman et Barker.
- L'identifiant relatif se traduit par une ligne pleine de l'association. Une association normale est sinon en points tillés.
- Les cardinalités sont : Absence de point terminal : (1,1) ;  
Point seul : (0,n) ; Point avec P : (1,n) ; Point avec Z : (0,1) ; Losange plein pour une association normale : (0,1)
- Héritage : une association avec un cercle médian
- Classification avec couverture :
  - deux traits au dessous du cercle
- Classification sans couverture :
  - un seul trait au dessous du cercle
- Classification avec disjonction :
  - un seul trait part du sur-type
- Classification sans disjonction :
  - plusieurs traits partent du sur-type

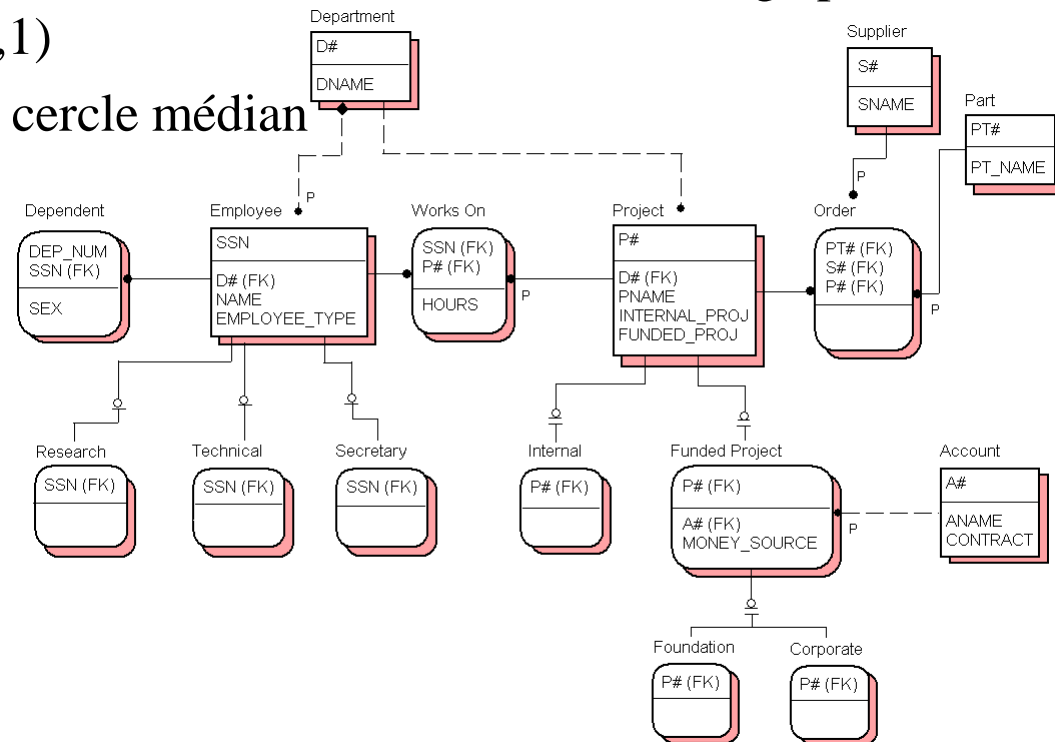
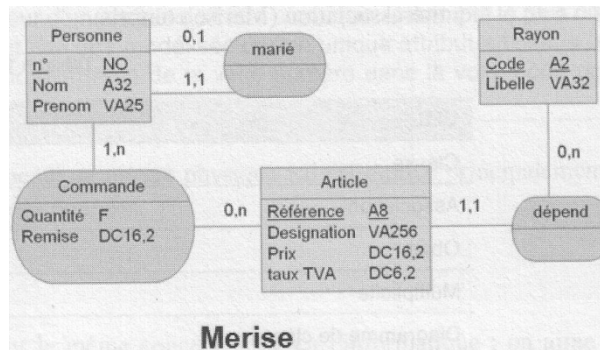


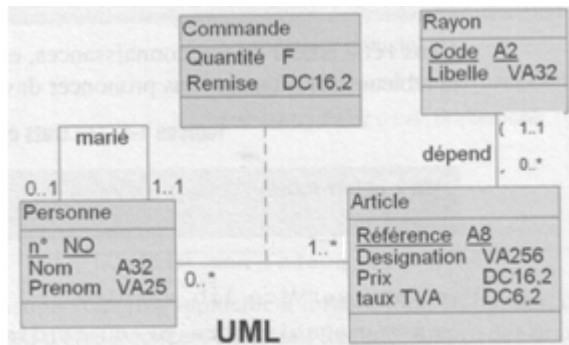
FIGURE 12: ERwin/ERX 2.0 Notation



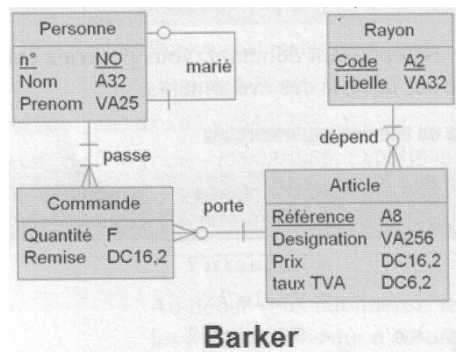
# Comparaison des notations



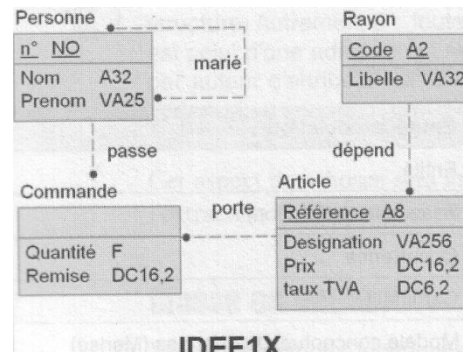
Merise



UML



Barker



IDEF1X

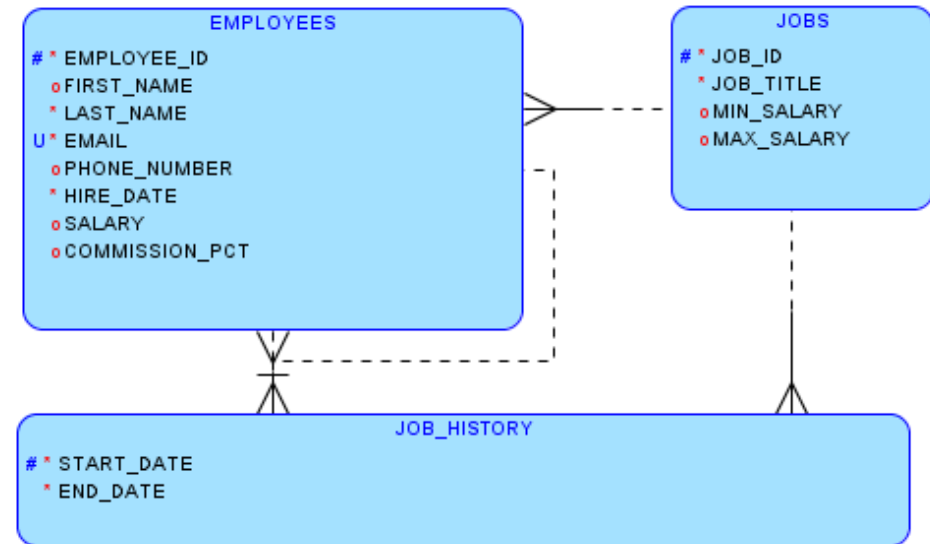
Catégorie	Notations
Notations seulement binaires versus les notations autorisant les associations ternaires	
Modèles n-aires	MERISE, UML, Chen
Modèles binaires	Barker, Bachman, IDEF1X, Information Engineering
Les cardinalités et les contraintes de participations peuvent être représentées par un couple (min,max)	
(min,max) Look here	MERISE
(min, max) Look Across	UML, IDEF1X, Information Engineering
Contraintes de participation Look here	Chen, Barker, Bachman
Contraintes de cardinalité Look Across	
Les clés étrangères sont –elles représentées au niveau du schéma conceptuel ou E-A étendu	
Clés étrangères absentes	MERISE, UML, Chen, Barker, Information Engineering
Clés étrangères modélisées	IDEF1X, Bachman.

# Oracle SQL Developer Data Modeler

- Oracle SQL Developer Data Modeler

permet de créer, importer ou exporter des modèles entité-Association (Entity Relationship Diagrams), des modèles relationnels (SGBD), mais aussi des structures de données au format DDL (Data Definition Language). L'outil ne se limite pas au seul support de la base Oracle mais s'interface aussi aux dictionnaires de données de DB2 et SQL Server. Il supporte aussi l'importation de catalogue (reverse engineering) depuis toute base compatible JDBC/ODBC ainsi que l'importation de fichiers XML générés par CA ERwin Data Modeler 4.X et de fichiers .VAR (Bachman Enterprise Model).

Michél Dubois



- Le niveau conceptuel (Logical Modeling) utilise les notations Bachman / Barker et prend en compte du OU EXCLUSIF et de l'héritage.
- Un "relational model" correspond à un modèle logique ou relationnel.
- Un "physical model" correspond à un modèle physique ORACLE, IBM DB2, MYSQL, MS SQL SERVER.

# JDBC & Oracle 10g

- Fichier bat :

```
@echo off
if "%OS%" == "Windows_NT" setlocal
set ORALIB=...
CLASSPATH=$CLASSPATH:$ORALIB/ojdbc14.jar:$ORALIB/orai18n.jar:$ORALIB/oc
rs12.zip
set TNS_ADMIN=C:\oracle\product\10.2.0\client_1\network\admin
set JAVA_PARAMS=-Xms512M -Xmx512M
@echo on
java %JAVA_PARAMS% -
    Djava.library.path=G:\vannes\prof\lcsd01\MD\win32\instantclient_10_2 -jar
    G:\vannes\prof\lcsd01\MD\lib\mapbuilder.jar
:end
```

Le pilote JDBC THIN 10g ne supporte pas l'authentification radius contrairement au pilote OCI et aux pilotes 11g OCI/THIN.

- Le fichier SQLNET.ORA du répertoire pointé par la variable d'environnement TNS\_ADMIN contient :

```
sqlnet.authentication.services=
radius
```

- L'URL complète JDBC est :

```
jdbc:oracle:oci:e0602180/1234@oraetud.univ-ubs.fr:1521:ORAETUD
```

A l'UBS, il faut utiliser le pilote OCI, le serveur Oracle 10g utilisant l'authentification radius :

```
java.util.Properties props = new
    java.util.Properties();
props.setProperty("sqlnet.authentication.services",
    "radius");
props.setProperty("user","e0602180");
props.setProperty("password","1234");
String url="jdbc:oracle:oci:"
    +"@oraetud.univ-ubs.fr:1521:ORAETUD";
conn = DriverManager.getConnection(url,props);
```

# Oracle SQL Developer 3

- La version de SQL Developer 3 est sortie :

[http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/in\[..\]](http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/in[..])

Les principales améliorations :

- support de Oracle Data Mining ;
- support de Oracle Spatial ;
- SQL Tuning Advisor ;
- SQL Developer Data Modeler 3.0 semble être intégré comme une extension de SQL Developer. Avant, c'était 2 logiciels différents.

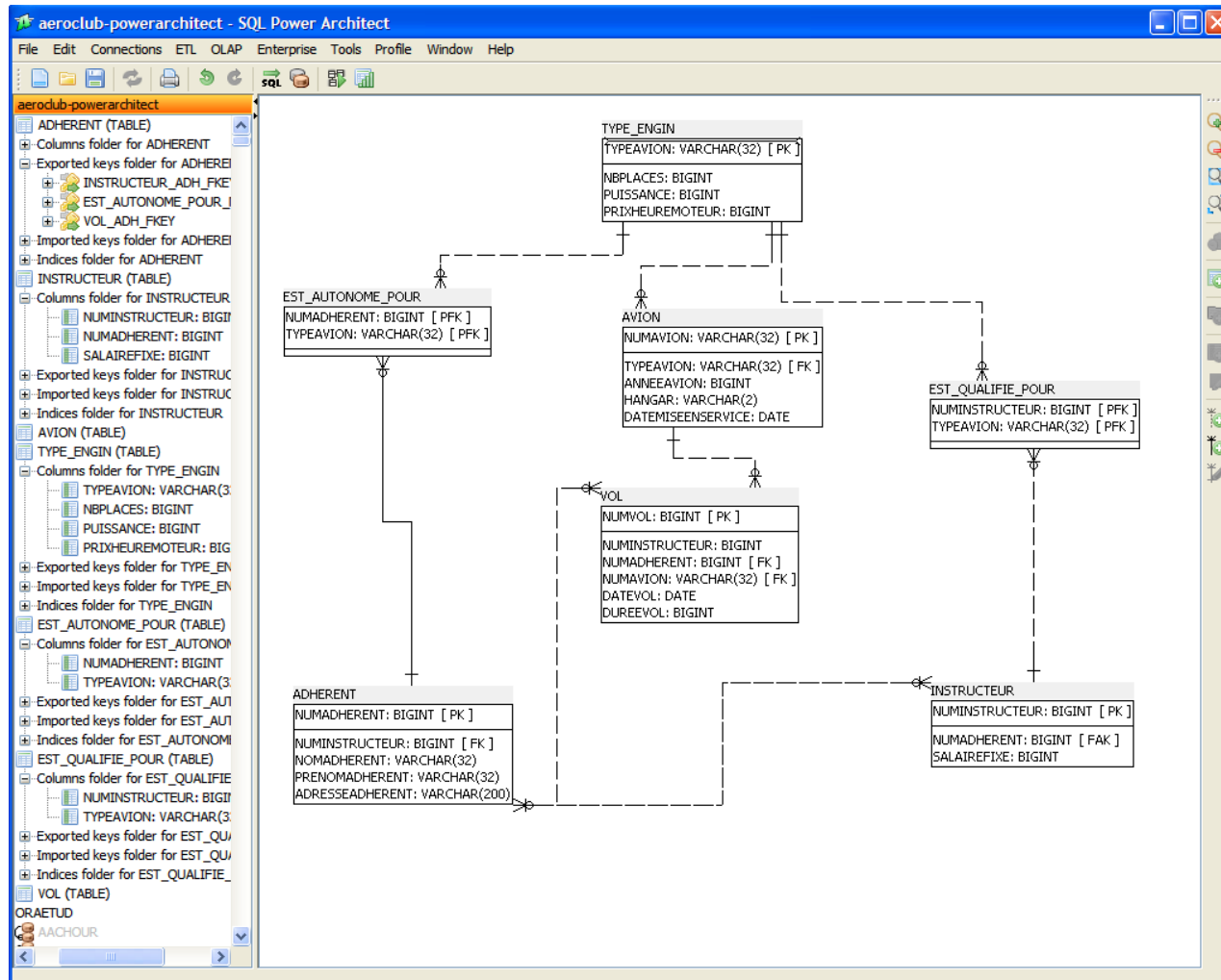
# SQL Power Architect

## AGL EER base de données

- SQL Power Architect 1.0.6 Community Edition (SQL Power Group Inc) peut être utilisé en enseignement.
- Edité par la compagnie québécoise SQL Power, Architect fait partie de leur gamme de logiciels orientés Business Intelligence (BI). Ce logiciel permet de modéliser une base et de générer automatiquement le schéma dans le SGBD de son choix (MySQL, PostgreSQL, Oracle...). Mais ce n'est pas tout, car on peut aussi créer les métadonnées pour les outils d'extraction, de transformation et de chargement (ETL) ou encore de concevoir un cube au format Mondrian. Il permet la comparaison de schémas (modèle actuel et la base actuelle), le forward engineering (génération de la base à partir du modèle), et le reverse engineering (génération du modèle à partir de la base),
- Ce logiciel est en java et utilise JDBC.

# SQL Power Architect

## AGL EER base de données



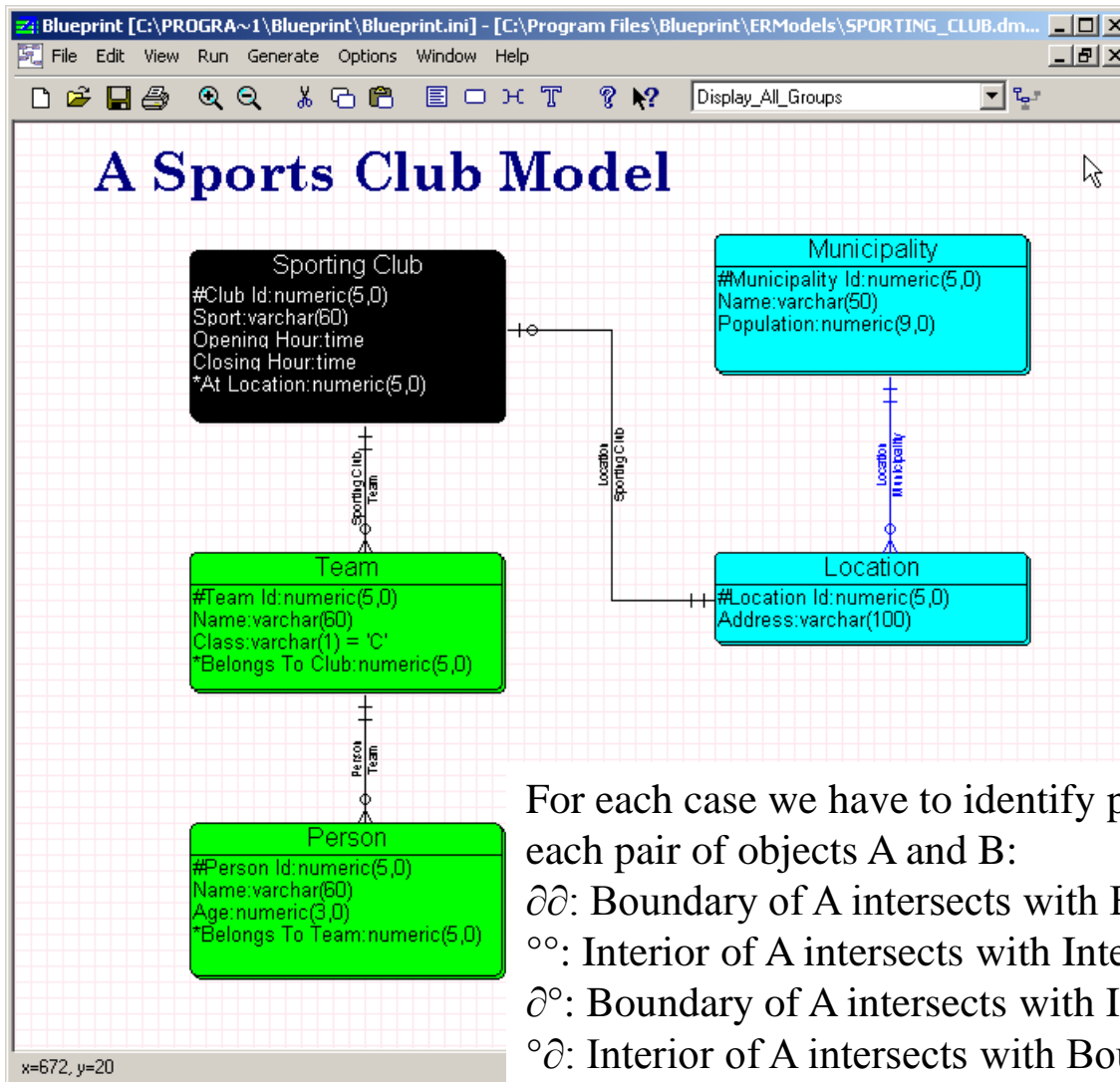
# Geometry Blueprint

## AGL EER/UML base de données

- [Geometry Blueprint](#) 3.0.1 supporte la modélisation UML 2, la notation objet en plus de la notation EER (Extended Entity-Relationship), non seulement l'application au spatial mais aussi les spécificités de la modélisation spatiale selon la méthode de Clementini (4-intersection modeling method). Elle peut déboucher sur de la génération de contraintes spatiales pour ORACLE, ESRI ARC/SDE à l'aide de templates... Ce qui ouvre la voie à la génération de code PostgreSQL/PostGIS.
- Il est gratuit et repose sur Java, JDBC.

# Geometry Blueprint

## AGL EER/UML base de données



$\partial\partial$	$\circ\circ$	$\partial^\circ$	$\circ\partial$	0-0	0-1	0-2	1-1	1-2	2-2
0 0 0 0	• •	•	•	□	□	□	□	□	□
1 0 0 0	•	→	→	□	□	□	□	□	□
0 1 0 0				□	□	□	□	□	□
1 1 0 0				□	□	□	□	□	□
0 0 1 0		→	→	□	□	□	□	□	□
1 0 1 0				□	□	□	□	□	□
0 1 1 0				□	□	□	□	□	□
1 1 1 0				□	□	□	□	□	□
0 0 0 1				□	□	□	□	□	□
1 0 0 1				□	□	□	□	□	□
0 1 0 1				□	□	□	□	□	□
1 1 0 1				□	□	□	□	□	□
0 0 1 1				□	□	□	□	□	□
1 0 1 1				□	□	□	□	□	□
0 1 1 1				□	□	□	□	□	□
1 1 1 1				□	□	□	□	□	□

For each case we have to identify possible intersections in four categories for each pair of objects A and B:

$\partial\partial$ : Boundary of A intersects with Boundary of B.

$\circ\circ$ : Interior of A intersects with Interior of B.

$\partial^\circ$ : Boundary of A intersects with Interior of B.

$\circ\partial$ : Interior of A intersects with Boundary of B.

where the  $\partial$  symbol denotes the boundary and the  $\circ$  symbol denotes the interior.



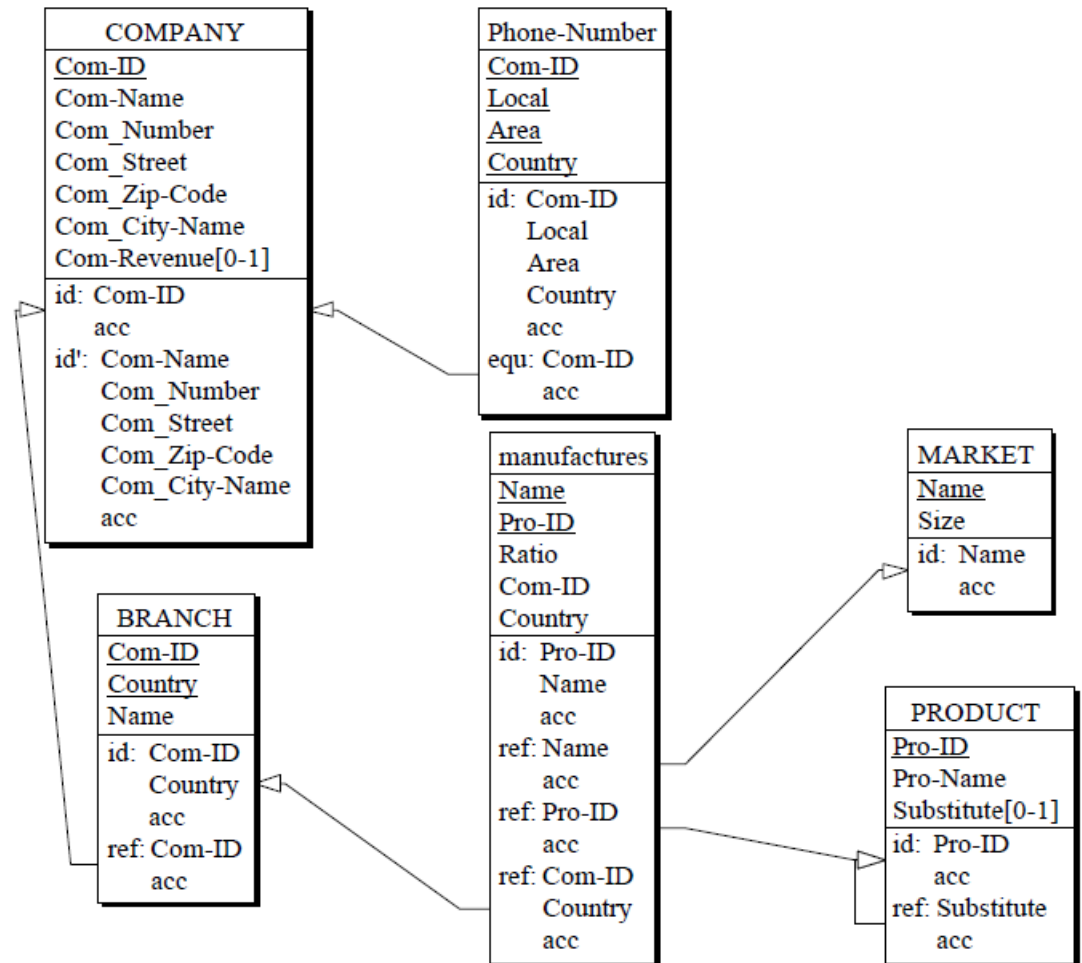
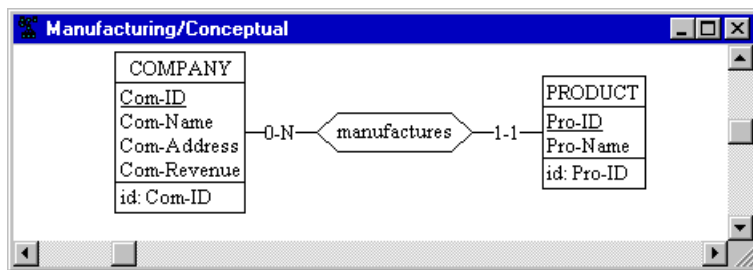
# DB-MAIN

## AGL EER/UML base de données IDM

- Si le modélisateur choisit de travailler au niveau conceptuel, DB-MAIN réalise automatiquement la transformation vers le niveau logique (relationnel) puis vers le niveau physique (SQL). Cette transformation automatique rentre dans le cadre de l'approche MDA qui prône la réalisation de systèmes en faisant abstraction de la plate-forme physique et de ses aspects technologiques. MDA introduit le PIM (*Platform Independent Model* ou *modèle indépendant de la plate-forme*) et le PSM (*Platform Specific Model* qui est le *modèle spécifique à la plate-forme*). Le passage de PIM à PSM doit se faire de façon automatisée ou semi automatisée.
- L'aspect MDA de DB-MAIN retient comme PIM, le modèle *objet* d'UML ou EER et comme PSM le modèle relationnel. DB-MAIN transforme les classes, les associations et les relations d'héritage.
- DB-MAIN présente d'autres caractéristiques comme la rétroconception ou reverse engineering (analyse d'un schéma au niveau physique), la migration, l'intégration de bases de données ou la prise en compte de XML.

# DB-MAIN

## AGL EER/UML base de données IDM



# MOSkitt

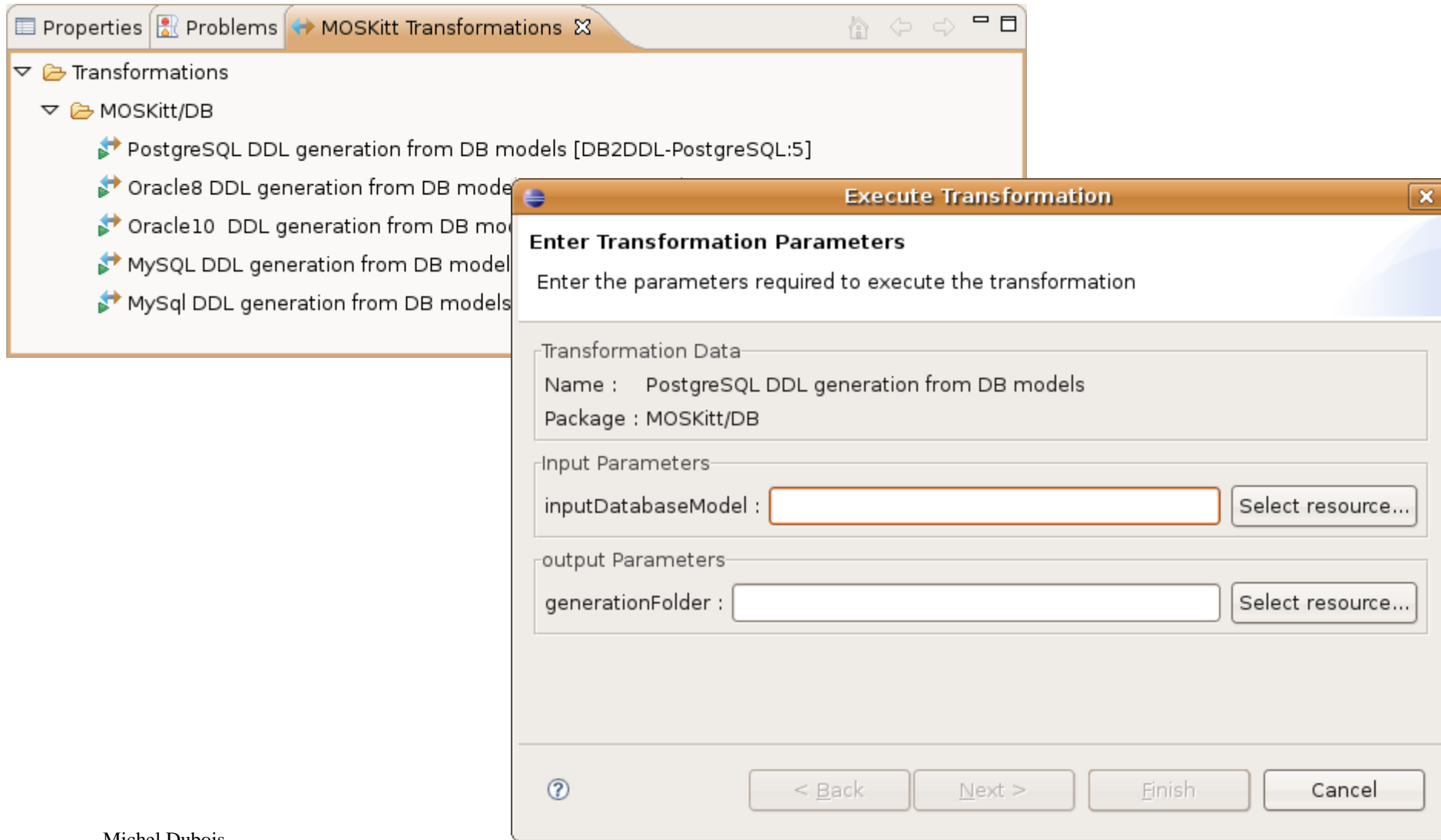
## AGL EER/UML base de données IDM

- MOSKitt est un outil open-source pour modélisation UML de bases de données par application de l'ingénierie dirigée par les modèles. Il est basé sur l'environnement open-source Eclipse RCP, EMF, GMF, GEF, ATL, Xtend, XPand2, AMW, UML2Tools, BPMN STP, Data Tools Platform. Il est donc plus qu'un outils base de données puisqu'il a en plus du modeleur UML, un modeleur BPMN (transformation vers UML), un éditeur d'interface graphique (transformation UML vers IHM). Il y a possibilité de transformer UML vers JPA (*Java Persistence API*) pour générer le code du ORM (object relational mapping) pour gérer la persistance des objets entre deux exécutions de l'application. L'IHM générée peut être en Java ou en PHP.
- Le modeleur UML2 supporte les profils UML et permet de faire un diagramme de classe, un diagramme de cas d'utilisation, un diagramme de séquence, un diagramme d'état-transition et un diagramme d'activités. Le moteur de génération de base de données permet de modéliser les tables, les clés, les vues, les utilisateurs, les rôles et les groupes ainsi que les Forward et Reverse engineering pour MySQL, PostgreSQL et Oracle.
- MOSKitt-Geo est un outil pour la conception de modèles de bases de données géospatiales et la création de schémas PostgreSQL/PostGIS et Oracle. Le modèle UML comprend des attributs géospatiaux. Une transformation automatique est appliquée pour générer un schéma vers le SGBDRO cible. Certaines features spatiales (points, multi-lignes, polygones) peuvent être appliquées, et un DDL (Data Definition Language) est généré pour créer une base de données PostGIS ou Oracle. De plus, vous pouvez également faire le reverse engineering de votre base de données spatiales et la refondre ou l'améliorer avec les outils graphiques de MOSKitt.

Michel Dubois

# MOSkitt

## AGL EER/UML base de données IDM



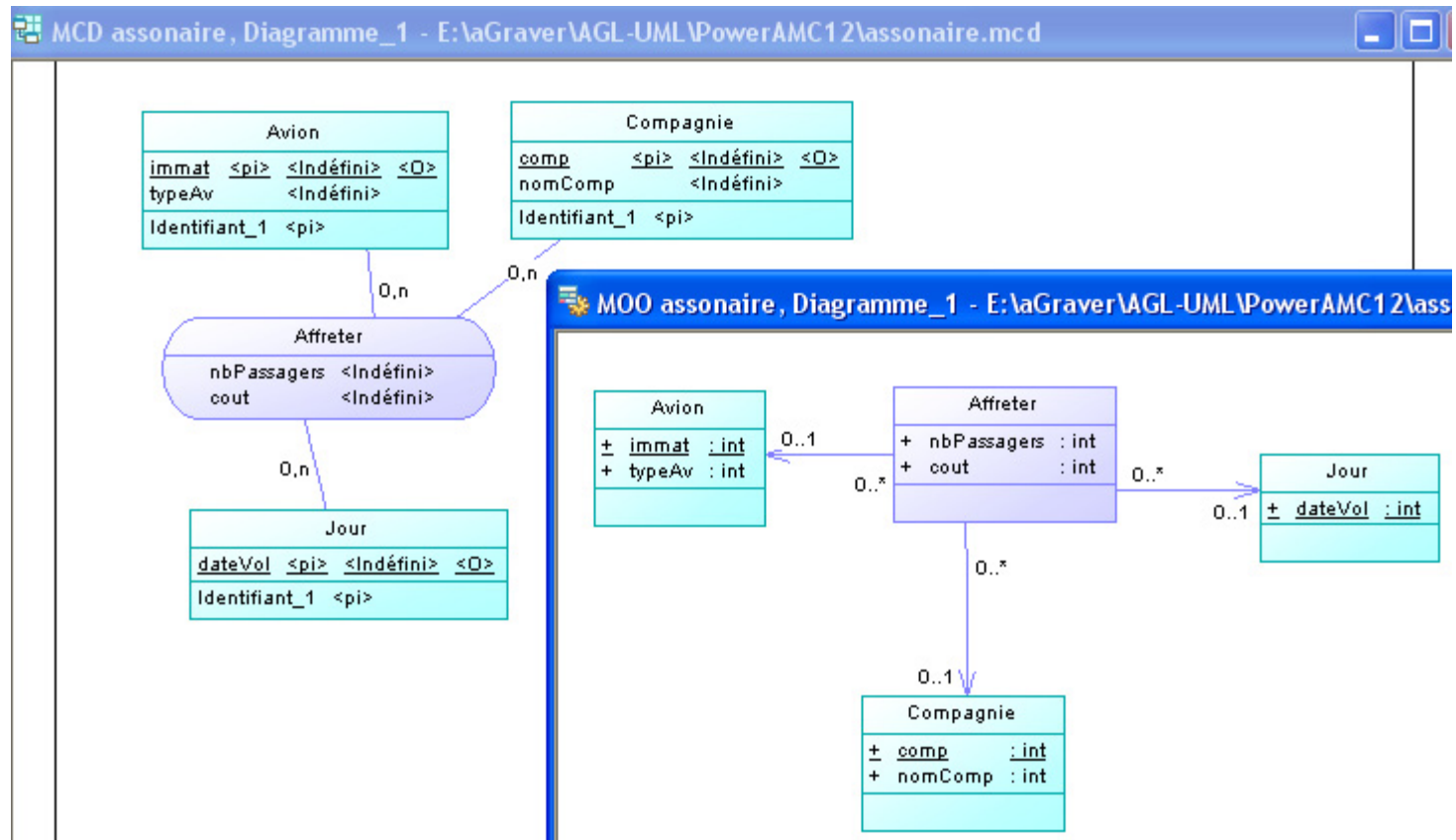
# PowerAMC

## AGL UML base de données

- PowerAMC (anciennement AMC\*Designor) est la version française de l'outil de modélisation PowerDesigner de Sybase. Concernant les bases de données, l'outil prend en charge trois types de modèles qu'on peut transformer entre eux :
  - le modèle conceptuel de données (MCD) qu'on pourra construire avec la notation Merise,
  - entité-relation ou IDEF1X ;
  - le modèle physique de données (MPD) qui correspond au niveau logique ;
  - le modèle orienté objet (MOO) au formalisme UML.
- Quelques points forts sont à mettre en exergue : un grand choix de SGBD est pris en compte, la robustesse et la qualité de l'ergonomie qui permet de travailler facilement avec différents diagrammes dans le même environnement. Les seuls points faibles se réduisent à la non-prise en compte des contraintes et des associations *n-aires avec la notation UML*.

# PowerAMC

## AGL UML base de données



# Win'Design

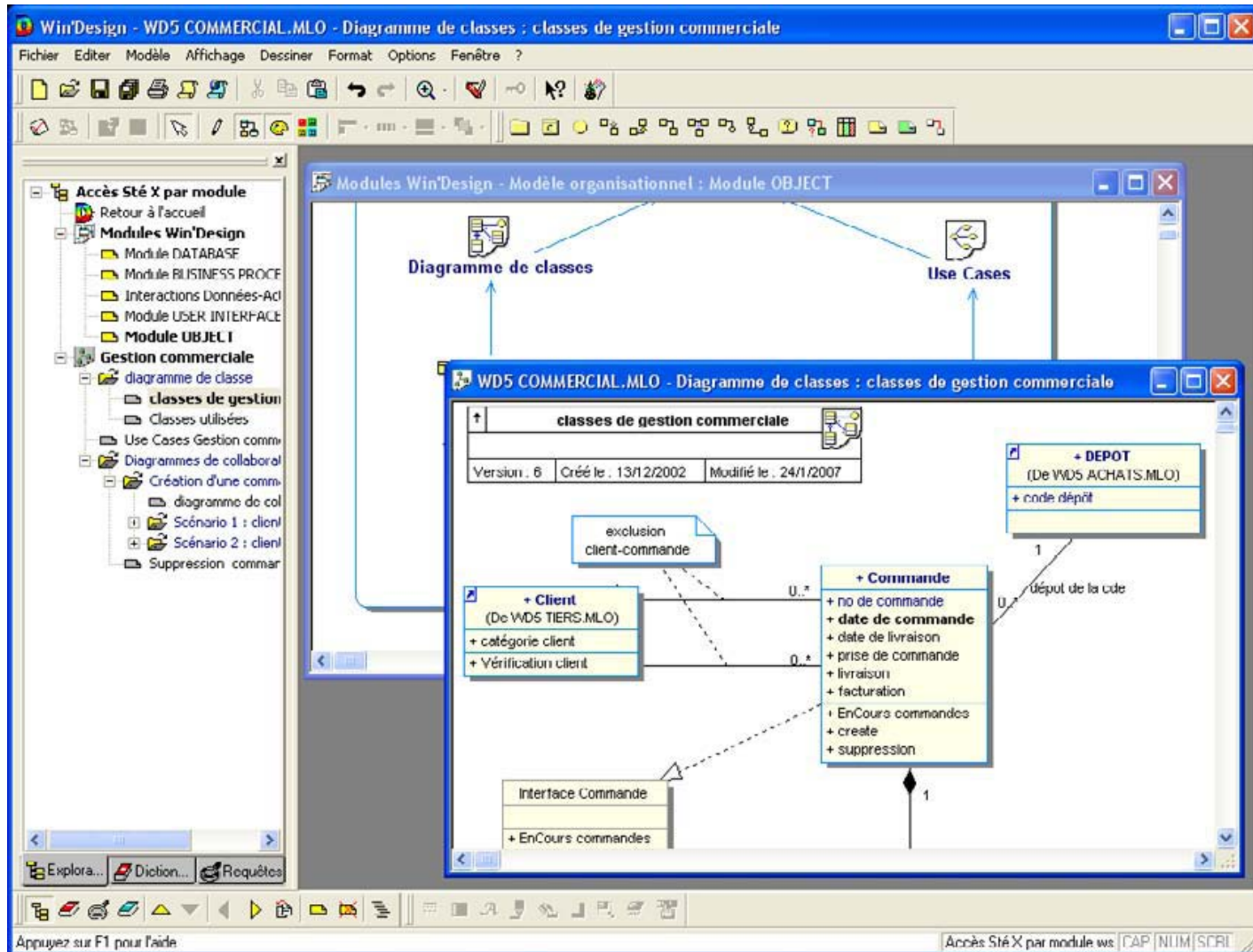
## AGL UML base de données

- Win'Design est le produit de la société CECIMA basée à Aix en Provence. Il est présent sur le marché français depuis 1995. Développé initialement pour Merise/2, la notation UML arrive en 2002 avec la version 5. Depuis l'outil est en évolution constante.
- La Gamme comprend quatre modules autonomes et complémentaires, qui s'articulent autour d'un référentiel (*Database* pour la conception et le reverse des bases de données, *Business Process* pour la modélisation des processus métier, *Object* pour la modélisation UML et *UserInterface* pour le maquetage des IHM). Vous devrez disposer du premier et du troisième module pour traduire des diagrammes de classes en script SQL. Comme PowerAMC, l'outil permet la double notation Merise/2 et UML 2 (sans le mode mixte de PowerAMC qui peut porter à confusion). Cet outil est le plus complet en ce qui concerne les contraintes Merise/2.
- Win'Design est probablement l'outil le plus facile à prendre en main et son interface est très intuitive. Il est le seul à traiter correctement la transformation des associations *n-aires* et qu'avec PowerAMC et Rational, il est capable d'extraire un modèle conceptuel sans connexion à la base (à partir du seul script SQL de création des tables et contraintes).



# Win'Design

## AGL UML base de données





# Visual Paradigm

- L'édition Entreprise de Visual Paradigm déclare des fonctionnalités relatives à SQL.
- Une fois saisi le diagramme de classes UML, nommez impérativement tous les rôles des liens d'associations (sinon la génération au niveau logique ne sera pas possible). Ensuite configurer votre transformation Tools/Object...(ORM)/Wizards... Rendez persistantes vos classes, choisissez un identifiant pour chacune et paramétrez votre connexion à la base de données cible. Une fois le modèle logique généré, il est nécessaire de le synchroniser Tools/Object...(ORM)/Synchronize... avec le diagramme de classes avant de pouvoir générer un script SQL.
- De toutes façons, vous devrez souvent agir sur le modèle logique (surtout pour les identifiants des classes-associations) avant de générer une base de données.
- Mais il n'a pas de support des identifiants de classes même si le support des association n-aire est apparu. On ne peut passer à SQL qu'à partir d'un modèle entité-relation. A partir de ce dernier, on peut générer un diagramme de classes.
- A l'IUT, la version d'évaluation académique n'est pas Entreprise.

# Visual Paradigm Academic Edition

Visual Paradigm Standard Edition est disponible pour les étudiants sur les postes windows du réseau ENSEIGNEMENT:

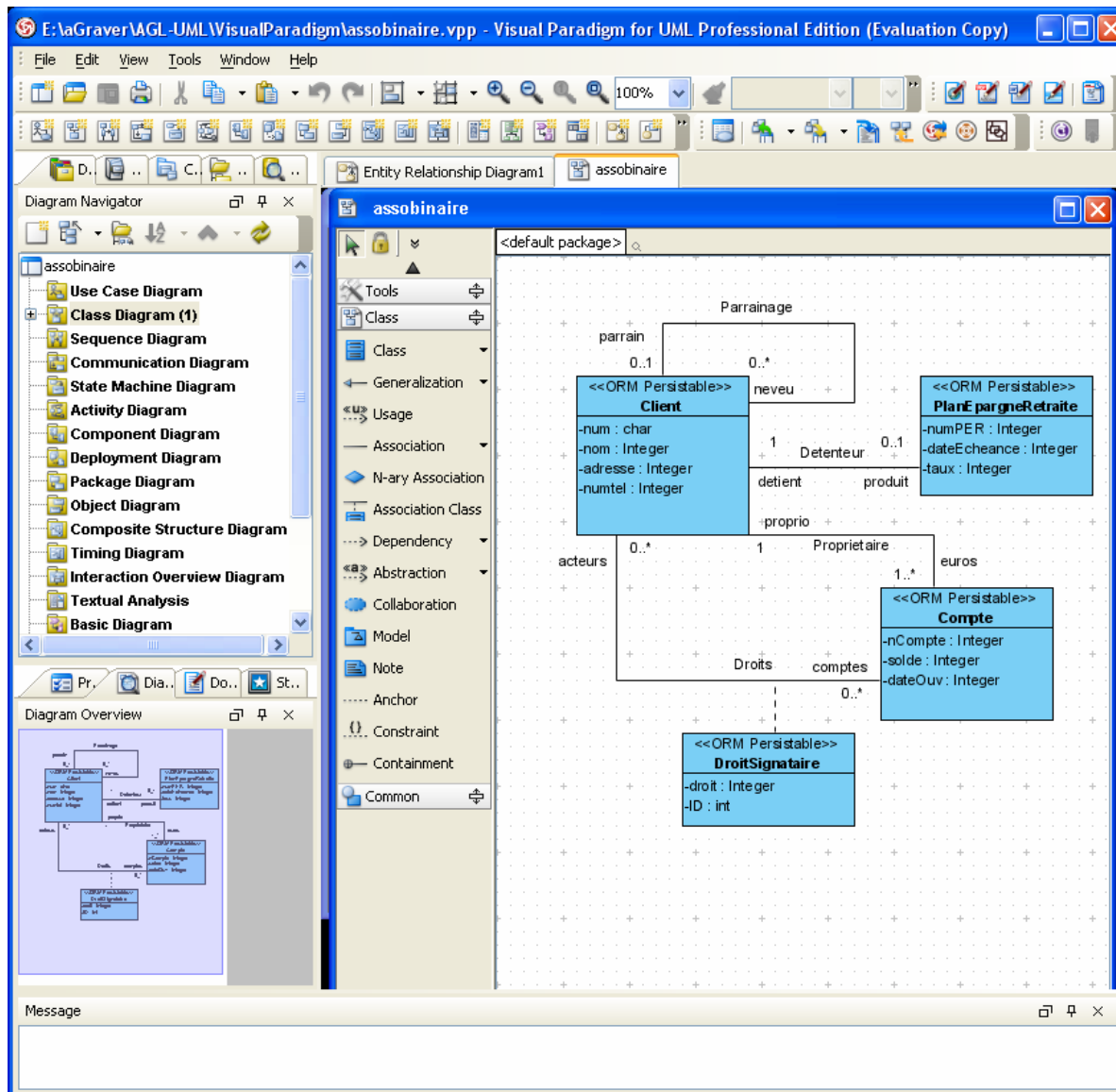
- génère depuis un diagramme de classe UML du code Java, C++, .Net (C#.NET, VB.NET et C++.NET), PHP 5.0, Python, Objective-C, et fait du reverse et de la synchronisation.

- génère depuis un diagramme de classe UML un schéma Entité-association étendu et les synchronise. A partir du diagramme Entité, association, VP génère le modèle logique puis les modèles physiques Oracle, MySQL, Microsoft SQL Server, HSQL, PostgreSQL, Derby, Firebird, SQLite. Fait du reverse Engineering.

- génère depuis un diagramme de classe UML un diagramme ORM et les synchronise. A partir du diagramme ORM, VP génère le code en Java pour Hibernate avec hibernate XML ou annotations JPA pour les bases de données cibles Oracle, MySQL, Microsoft SQL Server, HSQL, PostgreSQL, Derby, Firebird, SQLite. A partir du diagramme ORM, VP génère le code en PHP 5.0 pour Doctrine et EZPDO pour les bases de données cibles Oracle, MySQL, Microsoft SQL Server, PostgreSQL. A partir du diagramme ORM, VP génère le code en C# pour NHibernate pour les bases de données cibles Oracle, MySQL, Microsoft SQL Server, PostgreSQL, Firebird, SQLite.

Enfin VP est intégrable à Eclipse et Visual Studio.

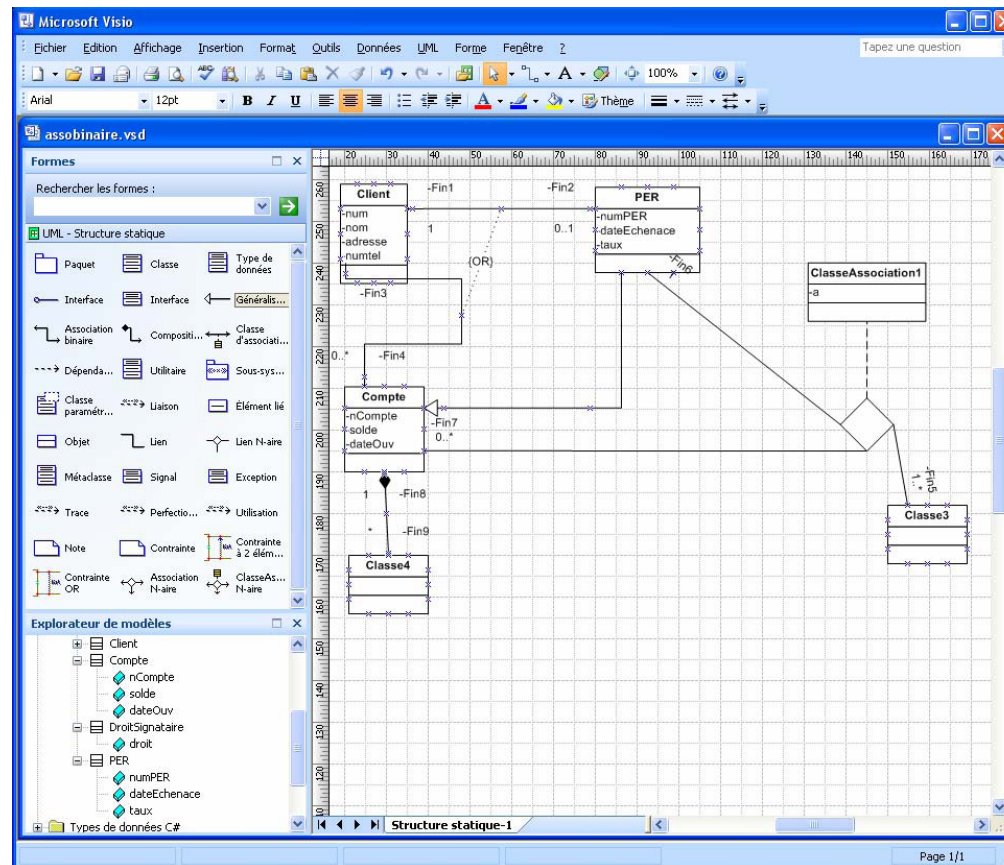
# Visual Paradigm



# Microsoft Visio

- Gamme de produit qui va du simple diagrammeur (Standard) à un véritable outil de modélisation de base de données et pour la conception de programmes (entreprise). Visio fait partie de la suite Office de Microsoft. Fournit un environnement de développement Visual Basic pour application ce qui permet de le personnaliser facilement.
- Support de nombreux formalismes (ER-Chen, Express-G, Martin ERD, ...). Support du reverse-engineering, génération de rapports. Supporte tous les diagrammes UML.
- En utilisant la notation UML, il n'est pas possible de générer de modèle ni de code. Pour ce faire, vous devrez travailler soit avec le modèle ORM (modèle conceptuel binaire graphique), soit avec IDEF1X (modèle relationnel graphique). Ce sont les seuls formalismes qui permettent de se connecter à une base. Dans ces deux cas, vous devrez utiliser Visio au sein de l'environnement de Visual Studio.
- Disponible avec l'offre MSDNAA en version Entreprise.

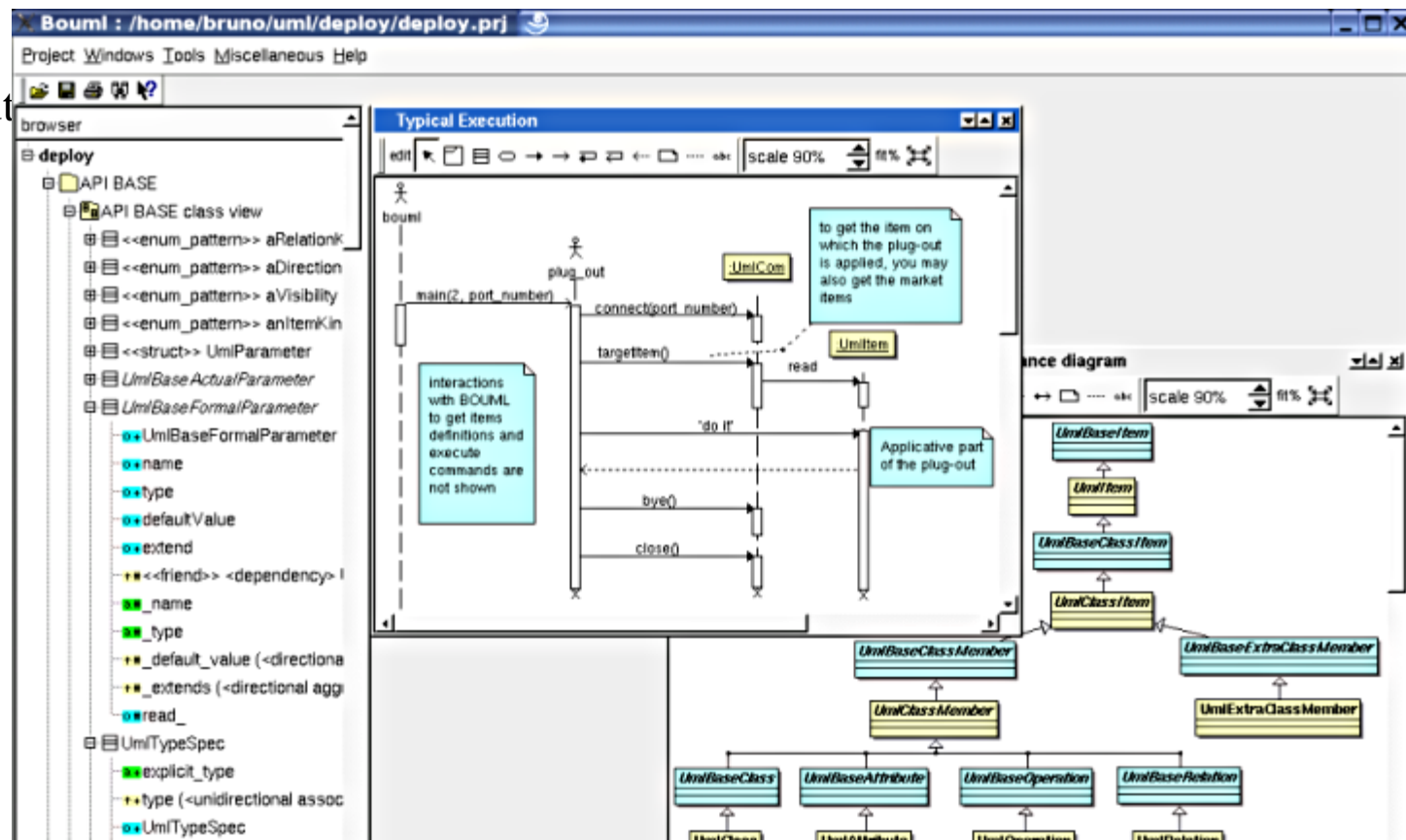
# Microsoft Visio



# BOUML

## AGL UML

- **BOUML** est une suite d'outils **UML 2** libres et **gratuits** dont un modeleur, vous permettant de spécifier et générer du code **C++**, **Java**, **Idl**, **Php** et **Python**.
- **BOUML** est disponible sous **Unix/Linux/Solaris**, **MacOS X** (Power PC et Intel) et **Windows**.
- Générateur XMI 1.2 et 2.1
- Import XMI 2.1
- Depuis 2012, il faut une licence ! (plus gratuit ☹)





# Chapitre 8

## Rappels sur la conception des bases de données : Modélisation directe

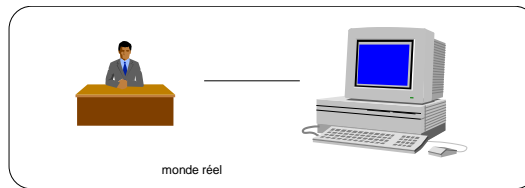
Michel Dubois

I.U.T. de Vannes

*Michel.Dubois@univ-ubs.fr*

[Retour au plan](#)

*DEMARCHE*



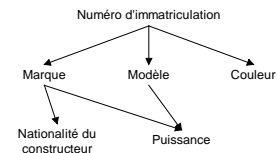
*DIRECTE*

PERCEPTION



**MODELISATION CONCEPTUELLE**

DICTIONNAIRE DES DONNEES  
GRAPHES DES DEPENDANCES FONCTIONNELLE  
OU  
NORMALISATION A PARTIR DE LA RELATION UNIVERSELLE



**MODELISATION LOGIQUE**

ACTEUR (no-acteur, ...)  
FILM (no-film, ...)  
JOUER (no-acteur, no-film)



**UTILISATION D'UN SGBDR**

Create Table ACTEUR...  
Create Index...





# *TERMINOLOGIE*

table  $\Leftrightarrow$  relation

enregistrement  $\Leftrightarrow$  occurrence  $\Leftrightarrow$  ligne de table  
 $\Leftrightarrow$  n-uplet  $\Leftrightarrow$  tuple

champ  $\Leftrightarrow$  attribut  $\Leftrightarrow$  colonne de table  $\Leftrightarrow$  donnée

intention d 'une relation  $\Leftrightarrow$  schéma d 'une relation

extension d 'une relation  $\Leftrightarrow$  contenu d 'une relation

## *APPLICATION : TABLE PERSONNE.*

Exemple - Soit une relation décrivant les caractéristiques d'une personne :

PERSONNE( Numéro S.S. : entier ; Nom : texte ; Prénom : texte ; Adresse : texte)

Une relation peut également être représentée "en extension" sous la forme de **table**.

Numéro S.S.	NOM	Prénom	Adresse
1700575124210	Martin	Pierre	10 rue des fleurs ...
1610930178125	Durand	Paul	2 Place des Lilas ...
1821113145312	Dupont	Jacques	3 Bd des maréchaux ...

# *CONSTRAINTES DU MODÈLE RELATIONNEL*

- Chaque enregistrement mémorise le même nombre de faits *Numéro S.S., Nom, Prénom, Adresse*.
- Chaque attribut contient le même type de faits sur chaque ligne et chacun de ces faits satisfait la définition de son domaine.
- Il ne peut y avoir qu'une seule valeur à l'intersection de chaque ligne et de chaque colonne. Ainsi, le premier individu (*Pierre Martin*) ne peut avoir qu'une seule adresse (*10 rue des fleurs*). Si on devait mémoriser en outre une résidence secondaire, nous serions obligés de rajouter un nouvel attribut à la table.

# *CONSTRAINTES DU MODÈLE RELATIONNEL*

- Il ne peut exister deux enregistrements exactement identiques, ce qui implique que l'ordre des lignes est sans importance.
- Deux attributs ne peuvent se partager le même nom, ce qui a pour conséquence que l'ordre d'apparition des attributs est également sans importance. Pour accéder à un attribut d'une table, il suffit de connaître son nom et non sa position.
- Une base de données contiendra généralement plusieurs tables. Dans cet ensemble, chaque table sera identifiée par un nom unique.

# *DD DE LA BASE LOCATION DE VOITURE*

CODE	LIBELLE	NATURE	CATEGORIE	DOMAINE	COMMENTAIRE
NUM_IMMA	Numéro d'immatriculation	Non calculée / saisie	Stable	texte[20] obligatoire	Identifiant de voiture
MARQUE	Marque	Non calculée / saisie	Stable	Texte[30] obligatoire	
MODELE	Modèle	Non calculée / saisie	Stable	Texte[30] obligatoire	
COULEUR	Couleur du véhicule	Non calculée / saisie	Evolutive	Texte[10] dans ( « Bleu », « Rouge », « Vert », « Noire »)	
NAT_CONS	Nationalité du constructeur	Non calculée / saisie	Stable	texte[20] obligatoire	
PUISSANCE	Puissance du moteur	Non calculée / saisie	Stable	Numérique Obligatoire	

# *ANOMALIES DE MANIPULATION*

Table Location de voitures par des clients							
N°Client	Nom Client	Prénom Client	N° Immatriculation	Marque voiture	Modèle voiture	Date de location	Nb jours de location
01	Martin	Pierre	91253 ABC 75	Renault	Twingo S	05/01/1997	1
01	Martin	Pierre	56465 CDE 78	FIAT	Panda AM 96	04/03/1997	3
02	Durand	Paul	91253 ABC 75	Renault	Twingo S	15/02/1997	2
02	Durand	Paul	1256 DCE 75	GMC	S 280	25/02/1997	4
02	Durand	Paul	8953 XY 56	Mercedes	D 250	16/04/1997	5
03	Dupont	Jacques	56465 CDE 78	FIAT	Panda AM 96	12/06/1997	2

- On ne peut ajouter un véhicule à la base que s'il est loué par un client.
- La suppression de l'unique location d'un véhicule peut entraîner la disparition des informations le concernant.
- La redondance sur la Fiat Panda AM 96 peut poser des problèmes de cohérence.

# *ANOMALIES DE MANIPULATION*

Table Voitures					
Numéro d'immatriculation	Marque	Modèle	Puissance	Nationalité du constructeur	Couleur
91253 ABC 75	Renault	Twingo S	6 CV	Française	Bleu
56465 CDE 78	FIAT	Panda AM 96	5 CV	Italienne	Rouge
132 YZ 01	Mercedes	S 280	12 CV	Allemande	Gris métallisé
1256 DCE 75	GMC	S 280	8 CV	Américaine	Azur
8953 XY 56	Mercedes	D 250	10 CV	Allemande	Rouge
4565 TD 34	FIAT	Punto	6 CV	Italienne	Noir
7897 BDE 75	Renault	Twingo S	6 CV	Française	Beige
2536 BRF 91	Mercedes	S 280	12 CV	Allemande	Bleu

- On ne peut introduire le fait que la nationalité dépend de la marque.
- La suppression de 1256DCE 75 entraîne la perte d'information disant que G M C est une compagnie américaine.
- La fusion de constructeurs automobiles peut entraîner un changement de nationalité. Cohérence!

# *LA NOTION DE DÉPENDANCE FONCTIONNELLE*

- **On dit qu'un attribut ou un groupe d'attributs B dépend fonctionnellement d'un attribut ou d'un groupe d'attributs A si à tout instant, chaque valeur de A n'a qu'une valeur associée B.**
- A est alors appelé la source de la dépendance fonctionnelle et B le but de la dépendance fonctionnelle.
- On note cette dépendance fonctionnelle :  $A \rightarrow B$
- $A \rightarrow B$  peut se lire de deux manières équivalentes :
  - *A détermine B*
  - *B dépend de A*
- Quand on connaît A, on connaît aussi B !
- Pour A connu, B peut être manquant dans le cas d'une DF faible.



## *APPLICATION : TABLE PERSONNE.*

Exemple - Soit une relation décrivant les caractéristiques d'une personne :

PERSONNE( Numéro S.S. : entier ; Nom : texte ; Prénom : texte ; Adresse : texte)

Une relation peut également être représentée "en extension" sous la forme de **table**.

Numéro S.S.	NOM	Prénom	Adresse
1700575124210	Martin	Pierre	10 rue des fleurs ...
1610930178125	Durand	Paul	2 Place des Lilas ...
1821113145312	Dupont	Jacques	3 Bd des maréchaux ...

## *DF DE L 'EXEMPLE*

NUMERO S.S.	→ PRENOM
NUMERO S.S.	→ ADRESSE
NUMERO S.S., NOM	→ PRENOM
NUMERO S.S., NOM	→ ADRESSE
NUMERO S.S., PRENOM	→ NOM
NUMERO S.S., PRENOM	→ ADRESSE
NUMERO S.S., ADRESSE	→ NOM
NUMERO S.S., ADRESSE	→ PRENOM
NUMERO S.S., NOM, PRENOM	→ ADRESSE
NUMERO S.S., NOM, ADRESSE	→ PRENOM
NUMERO S.S., PRENOM, ADRESSE	→ NOM

# *PAR CONTRE CES DF SONT FAUSSES ...*

NOM	→ NUMERO S.S.
PRENOM	→ NUMERO S.S.
ADRESSE	→ NUMERO S.S.
NOM	→ PRENOM
PRENOM	→ NOM
ADRESSE	→ NOM
NOM	→ ADRESSE
PRENOM	→ ADRESSE
ADRESSE	→ PRENOM
NOM, PRENOM	→ NUMERO S.S.
NOM, PRENOM, ADRESSE	→ NUMERO S.S.

# *DEFINITION : DEPENDANCE FONCTIONNELLE*

- Soit  $U$  l'ensemble des attributs du domaine étudié,
- Soit  $V$  un sous ensemble de  $U$  :  $V \subseteq U$ ,
- Soit  $R(V)$  un schéma de relation,
- $r$  une relation de schéma  $R$ ,
- $X \subseteq V, Y \subseteq V$  deux sous-ensembles d'attributs de  $R$ .

La dépendance fonctionnelle  $X \rightarrow Y$  est vrai dans  $r$ ,  
ssi (si et seulement si)

tous les nuplets de  $r$  qui ont même valeur pour (tous) les attributs de  $X$ , ont même valeur pour (tous) les attributs de  $Y$ .

# *DEFINITION : DEPENDANCE FONCTIONNELLE*

« tous les nuplets de  $r$  qui ont même valeur pour (tous) les attributs de  $X$ , ont même valeur pour (tous) les attributs de  $Y$  »

LIVRAISON(no-bon, ref-art, date-bon, libellé-art, qté-art, code-famille, libellé-famille)

$F = \{ \text{no-bon} \rightarrow \text{date-bon}, \text{ref-art} \rightarrow \text{libellé-art}, \text{code-famille} \rightarrow \text{libellé-famille} \}$

## LIVRAISON

no-bon	date-bon	ref-art	libellé-art	qté-art	code-famille	libellé-famille
10	20-04-94	SDF0875	Poupée Barbie	200	JF	jouets filles
10	20-04-94	RMI7543	Monopoly	250	JS	jeu de société
18	25-04-94	RMI7543	Monopoly	20	JS	jeu de société
...	...	...	...	...	...	...

# *ENSEMBLE DE DF*

Soit  $U$  l'ensemble des attributs du domaine étudié,

Soit  $V$  un sous ensemble de  $U$  :  $V \subseteq U$ ,

Soit  $R(V)$  un schéma de relation,

Soit  $r$  une relation de schéma  $R$ ,

Soit  $F$  un ensemble de dépendances fonctionnelles qui sont vraies dans  $r$ ,

On dit qu'un ensemble  $F$  de dépendances fonctionnelles implique logiquement la dépendance fonctionnelle  $X \rightarrow Y$  si et seulement si tout contenu de  $r$  vérifiant  $F$  vérifie aussi  $X \rightarrow Y$ .

On note  $F \Rightarrow X \rightarrow Y$

Exemple :

$R(A,B,C)$  et  $F=\{A \rightarrow B, B \rightarrow C\}$  alors  $F \Rightarrow A \rightarrow C$

## *FERMETURE D'UN ENSEMBLE DE DF*

- La fermeture ou clôture ou couverture  $F^+$  d'un ensemble  $F$  de dépendances fonctionnelles est l'union de  $F$  et de l'ensemble des dépendances fonctionnelles logiquement impliquées par  $F$ .
- Il existe des propriétés des dépendances fonctionnelles qui permettent de déterminer  $F^+$ .
- Deux ensembles de dépendances fonctionnelles  $F$  et  $G$  sont équivalents si  $F^+ = G^+$ . Un ensemble de DF peut être plus facile à manipuler qu'un autre.

# *PROPRIÉTÉS DES DÉPENDANCES FONCTIONNELLES*

- Ce sont les Axiomes d'ARMSTRONG :
  - Reflexivité
  - Augmentation
  - Transitivité
  - Union, décomposition et pseudo-transitivité peuvent être déduites des autres axiomes.
- Ces axiomes permettent de déterminer les dépendances fonctionnelles logiquement impliquées par d 'autres.



# *LA TABLE VOITURE*

VOITURE					
Numéro d'immatriculation	Marque	Modèle	Puissance	Nationalité du constructeur	Couleur
91253 ABC 75	Renault	Twingo S	6 CV	Française	Bleu
56465 CDE 78	FIAT	Panda AM 96	5 CV	Italienne	Rouge
3456 FGR 75	GMC	D 350	12 CV	Américaine	Bleu
132 YZ 01	Mercedes	S 280	12 CV	Allemande	Gris métallisé
1256 DCE 75	GMC	S 280	8 CV	Américaine	Azur
8953 XY 56	Mercedes	D 250	10 CV	Allemande	Rouge

# *RÉFLEXIVITÉ*

$$B \subseteq A \Rightarrow A \rightarrow B$$

- Tout ensemble d'attributs détermine lui-même ou une partie de lui-même.
- Toute partie d'un ensemble d'attributs dépend de l'ensemble complet.

Numéro d'immatriculation, Marque  $\rightarrow$  Numéro d'immatriculation, Marque

Numéro d'immatriculation, Marque  $\rightarrow$  Marque

Numéro d'immatriculation, Marque  $\rightarrow$  Numéro d'immatriculation.

# *AUGMENTATION*

$A \rightarrow B \Rightarrow A, C \rightarrow B, C$

- Si A détermine B (B dépend de A), les deux ensembles d'attributs peuvent être enrichis par un même troisième dans la relation de dépendance.

Numéro d'immatriculation  $\rightarrow$  Marque  $\Rightarrow$

Numéro d'immatriculation, Couleur  $\rightarrow$  Marque, Couleur

# *TRANSITIVITÉ*

$$A \rightarrow B \text{ et } B \rightarrow C \Rightarrow A \rightarrow C$$

- Il y a dépendance fonctionnelle transitive entre 2 ensembles d'attributs  $A \rightarrow C$  lorsque A détermine un ensemble d'attributs B, qui lui-même détermine C.
- Il y a dépendance fonctionnelle transitive entre 2 attributs  $A \rightarrow C$  lorsque C dépend d'un attribut B, qui lui-même dépend de A.

Numéro d'immatriculation  $\rightarrow$  Marque et

Marque  $\rightarrow$  Nationalité du constructeur

$\Rightarrow$  Numéro d'immatriculation  $\rightarrow$  Nationalité du constructeur

# *UNION*

$A \rightarrow B$  et  $A \rightarrow C \Rightarrow A \rightarrow B, C$

- Lorsque qu'un attribut détermine deux groupes d'attributs distincts, il détermine leur association.

Numéro d'immatriculation  $\rightarrow$  Marque

et Numéro d'immatriculation  $\rightarrow$  Couleur

$\Rightarrow$

Numéro d'immatriculation  $\rightarrow$  Marque, Couleur

# *PSEUDO-TRANSITIVITÉ*

$$A \rightarrow B \text{ et } B, C \rightarrow D \Rightarrow A, C \rightarrow D$$

# *DÉCOMPOSITION*

$$A \rightarrow B \text{ et } C \subseteq B \Rightarrow A \rightarrow C$$

Numéro d'immatriculation  $\rightarrow$  Marque, Couleur

$\Rightarrow$

Numéro d'immatriculation  $\rightarrow$  Marque

et Numéro d'immatriculation  $\rightarrow$  Couleur

- Il y a diminution des buts : opération d 'éclatement
- Quand le but d 'une df est monoattribut, on dit que la df est canonique. L 'opération d 'éclatement permet de se ramener à des df canoniques

# *TYPOLOGIE DES DÉPENDANCES FONCTIONNELLES EN FONCTION DES AXIOMES D 'AMSTRONG*

- Par rapport à l'augmentation :  
Dépendances fonctionnelles élémentaires
- Par rapport à la transitivité :  
Dépendances fonctionnelles directes.



# *DÉPENDANCE FONCTIONNELLE ÉLÉMENTAIRE*

- On dit qu'une dépendance fonctionnelle est élémentaire si la source ne contient pas d'attributs superflus. La source ne contient pas d'attributs superflus si le fait d'enlever un attribut quelconque dans la source entraîne la disparition de la dépendance fonctionnelle. La df est optimisée vis à vis de la propriété d'augmentation.
- Bien entendu, toute dépendance fonctionnelle dont la source ne contient qu'un seul attribut est élémentaire.

$$\begin{array}{lll} A, B \xrightarrow{dfé} C & \text{si} & A, B \rightarrow C \\ & \text{et} & A \nrightarrow C \\ & & B \nrightarrow C \end{array}$$

# *DÉPENDANCE FONCTIONNELLE DIRECTE*

- Une dépendance fonctionnelle est directe s'il n'existe pas de transitivité entre la source et le but. Elle n'est pas reconstituable par transitivité. Elle est donc optimisée vis à vis de la propriété de transitivité.

$A \xrightarrow{dfd} B$  ssi il n'existe aucun ensemble d'attribut  $X$  tel que  $A \rightarrow X$  et  $X \rightarrow B$

# *EXEMPLES*

- Numéro d'immatriculation  $\xrightarrow{dféd}$  Couleur
- Numéro d'immatriculation, Marque  $\xrightarrow{dfd}$  Couleur
- Marque, Modèle  $\xrightarrow{dféd}$  Puissance
- Numéro d'immatriculation  $\xrightarrow{dfé}$  Nationalité du constructeur

# *COUVERTURE MINIMALE*

- De tout ensemble de dépendance fonctionnelle, on peut extraire au moins un ensemble de dépendances fonctionnelles élémentaires et directes équivalent : modulo la transitivité et l'augmentation.
- On l'appelle sa couverture minimale. Elle minimise l'expression des contraintes structurelles du S.I. que sont les DF.
- Donc tout ensemble de dépendances fonctionnelles admet une couverture minimale, en général non unique.

# *COUVERTURE MINIMALE*

La couverture minimale  $G$  d'un ensemble  $F$  de dépendances fonctionnelles (DF) est un ensemble de DF tel que :

1. On peut déduire de  $G$  les mêmes DF que de  $F$  :  $G^+ = F^+$
2. Il n'y a qu'un attribut à droite dans toutes les DF de  $G$  (décomposition)
3. Toutes les dépendances sont utiles : si on enlève une quelconque, on ne peut obtenir  $F^+$ .
4. Toutes les dépendances sont élémentaires.

# *NOTION DE CLÉ PRIMAIRE*

- Une table doit **obligatoirement** comporter une clé primaire.
- il faut que chaque enregistrement soit doté d'un identifiant pour être retrouvé : c'est le rôle de la clé primaire.
- Par convention, la clé primaire d'une table apparaît soulignée dans la représentation en intention d'une table.
- On distingue deux types de clés primaires :
  - Les clés primaires simples, constituées par un seul attribut.
  - Les clés primaires composées : parfois on définit une clé primaire formée de plusieurs attributs de manière à obtenir l'unicité de la valeur de la clé.

# *PROPRIÉTÉS DE LA CLÉ PRIMAIRE*

- Par définition, la clé primaire doit déterminer tous les regroupements possibles des autres attributs de la table.
- Elle doit être minimale.
- Elle ne peut avoir deux fois la même valeur pour deux tuples. Il n'y a pas de doublons. Il y a une contrainte unicité (UQ) sur toute la clé primaire.
- De préférence, une clé primaire est composée d'attributs stables dans le temps. Une modification de la clé primaire risque de poser de gros problèmes de cohérence, donc de maintenance de la base de données.
- Une clé primaire doit être systématiquement renseignée, donc on ne peut autoriser la présence de valeurs nulles dans une de ses composantes qui, dès qu'elles se multiplient, conduisent inévitablement à l'apparition de doublons. La contrainte (NN) sur chaque partie de la clé primaire.

# LES CLÉS CANDIDATES

- Au sein d'une table, plusieurs attributs ou groupes d'attributs peuvent remplir les conditions (unicité, minimalité, pas de valeurs nulles, stabilité) pour devenir clé primaire de la table : ils formeront les clés candidates.

$R(\text{Ville}, \text{Adresse}, \text{Code postal}) \quad V = \{ \text{Ville:chaîne}, \text{Adresse:chaîne}, \text{Code postal:entier} \}$

$F = \{ \text{Ville}, \text{Adresse} \rightarrow \text{Code postal}, \text{Code postal} \rightarrow \text{Ville} \}$

$\text{Ville}, \text{Adresse} \rightarrow \text{Code postal}, \text{Ville}, \text{Adresse}$

$\text{Code postal}, \text{Adresse} \rightarrow \text{Ville}, \text{Code postal}, \text{Adresse}$

- Dans ce cas on retiendra comme clé primaire la clé candidate la plus petite au niveau de la taille de stockage (Code postal, Adresse ). Toutes les clés candidates ont une contrainte d'unicité (UQ) sur leur globalité.
- Pour une relation R, il y a toujours une clé candidate.
- Au pire V tout entier peut être la clé candidate. Il vaut mieux envisager de créer un attribut incrémenté à chaque création d'une occurrence de la relation spécialement pour ce cas.



# *LES CLÉS ÉTRANGÈRES (OU CLÉS EXTERNES)*

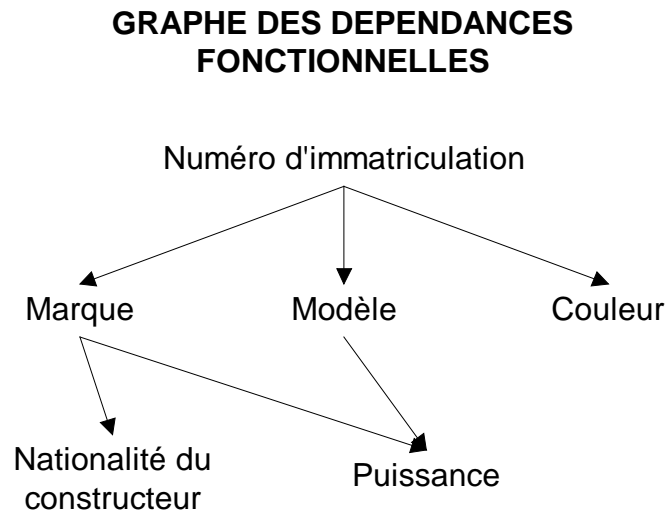
- Une clé étrangère est un attribut (clé étrangère simple) ou un groupe d'attributs (clé étrangère composée) d'une table dont les valeurs font référence à celles de la clé primaire d'une autre table.
- Les clés étrangères permettent ainsi de relier des tables à l'aide de la jointure naturelle.
- Les clés étrangères comportent généralement des doublons. Elle peut cependant avoir une contrainte d'unicité sur sa globalité (UQ).
- Elles autorisent généralement la présence de valeurs manquantes si elles n'entrent pas dans la composition de la clé primaire de leur table. Elles peuvent cependant avoir une contrainte d'obligation (NN) sur chacune de ses parties.
- Par contre, toute valeur renseignée de la clé étrangère doit être présente dans la clé primaire de la table de référence pour que le SGBDR puisse relier les deux tables.

# *LE GRAPHE DES DÉPENDANCES FONCTIONNELLES ÉLÉMENTAIRES ET DIRECTES OU DE COUVERTURE MINIMALE*

- Un graphe orienté dont les noeuds sont les attributs du dictionnaires de données
- Ces noeuds sont ensuite réunis par des flèches représentant les dépendances fonctionnelles élémentaires et directes.
- Ces flèches sont orientées de la source vers le but de la dépendance fonctionnelle mise en évidence.

# *EXEMPLE DE GRAPHE*

- Pour nos seules données relatives aux voitures :

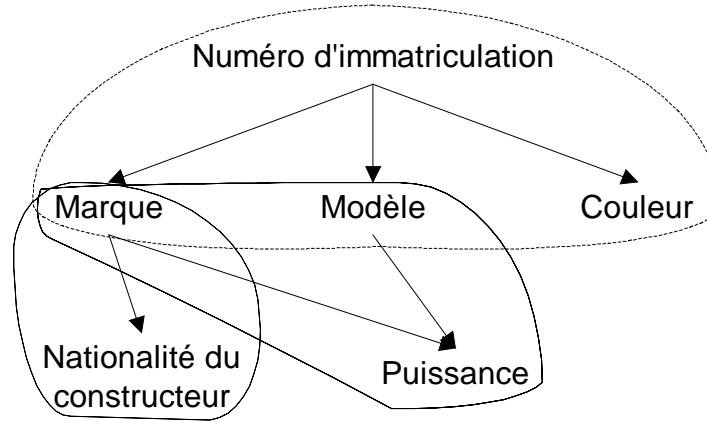


# *GRAPHES DE DÉPENDANCES ET FORMES NORMALES*

- Un graphe de couverture minimale des dépendances fonctionnelles donne immédiatement un schéma de tables en troisième forme normale.
- Pour cela, il suffit de déterminer des ensembles de données qui dépendent toutes fonctionnellement de la même donnée ou du même groupe de données. Chaque ensemble donnera une table dont la clé primaire sera la source de la dépendance fonctionnelle commune.
- Les données appartenant à plusieurs ensembles seront nécessairement un composant de clé primaire dans une table et/ou une clé étrangère dans une autre.

# EXEMPLE

## GRAPHE DES DEPENDANCES FONCTIONNELLES



Ce graphe donnera les tables suivantes :

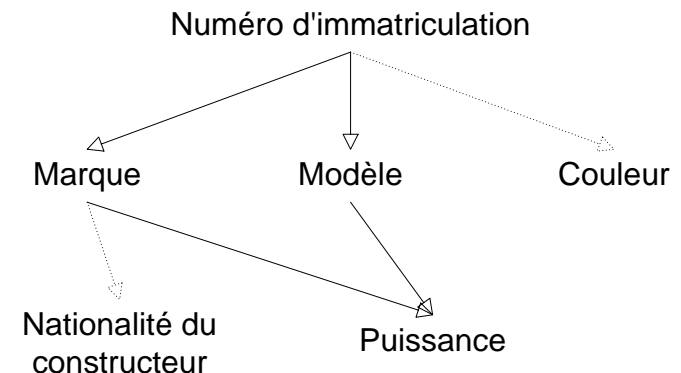
- Voiture (Numéro d'immatriculation, #[Marque, Modèle], Couleur)
- Nationalité (Marque, Nationalité du constructeur)
- Puissance (#Marque, Modèle, Puissance)

# *DÉPENDANCES FONCTIONNELLES FAIBLES/FORTES*

- Une dépendance fonctionnelle est forte si et seulement si son but est toujours valué. Dans ce cas, il y a une contrainte d'obligation de présence sur le but.
- Une dépendance fonctionnelle est faible si et seulement si son but peut ne pas être valué. C'est la situation par défaut.
- Une voiture peut ne pas avoir de couleur. Par contre toute voiture à une marque.

Numéro d'immatriculation      ----->      Couleur

Numéro d'immatriculation      —————>      Marque



## *D.F. INTER-RELATIONS*

- Une clé étrangère exprime une DF inter-relation : quand on connaît la clé étrangère, on connaît la valeur de la clé primaire référencée.
- Exemple :

$R(\underline{A}, B, C) \quad R'(\underline{D}, \#A)$

Les dépendances exprimées par ce schéma de base :

$$F = \{ A \rightarrow B, \quad A \rightarrow C, \quad D \rightarrow A \}$$

mais on a aussi  $R'.A \rightarrow R.A$  si on tient compte du schéma relationnel.

Cette DF ne représente pas une contrainte du Système d'Information.

Elle est plutôt d'ordre technique et ne se rencontre pas lors de l'exploitation de la base de données : la raison d'être des clés étrangères est de permettre la recomposition par jointure naturelle. Or cette dernière élimine l'attribut doublon dans la relation résultat !

# *SCHEMA RELATIONNEL ET DF*

- Une dépendance fonctionnelle est une propriété du Système d'Information qui est indépendante du découpage relationnel, puisque celui-ci n'existe pas à ce moment.
- Ce qui n'empêche pas de revenir sur les dépendances fonctionnelles après découpage (ou décomposition) et de repérer des dépendances fonctionnelles qui sont devenues inter relation dans ce contexte.
- Retour à l'exemple :

$F = \{A \rightarrow B, A \rightarrow C, D \rightarrow A\}$  et  $R(\underline{A}, B, C) \quad R'(\underline{D}, \#A)$

On a  $R'.D \rightarrow R'.A$  et  $R'.A \rightarrow R.A$  donc  $R'.D \rightarrow R.A$

$R.A \rightarrow R.B$  est intra-relation.

$R.A \rightarrow R.C$  est intra-relation.

$R'.D \rightarrow R.A$  est inter-relation.



# GRAPHE DES D.F. INTER-RELATIONS

Voiture.Numéro d'immatriculation  $\rightarrow$  Puissance.Marque,

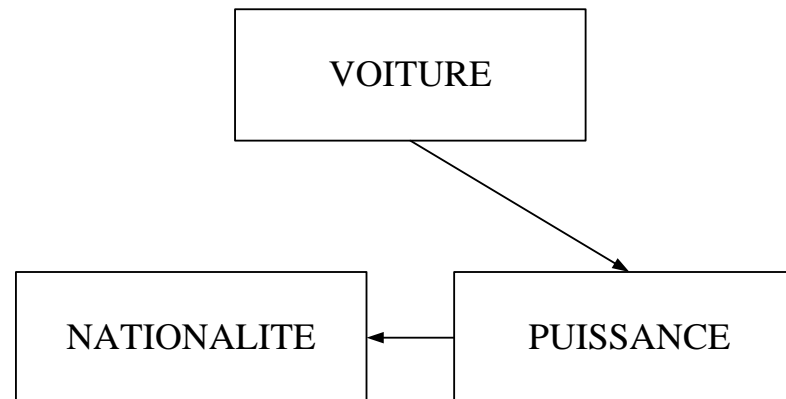
Voiture.Numéro d'immatriculation  $\rightarrow$  Puissance.Modele,

*donc*

Voiture.Numéro d'immatriculation  $\rightarrow$  Puissance.[Marque, Modele]

Puissance.[Marque, Modele]  $\rightarrow$  Nationalite.Marque }

Tout véhicule a une marque et un modèle. De plus toute puissance dépend d'une marque. Donc les df inter-relations sont toutes fortes.



# *DÉPENDANCES FONCTIONNELLES INTER-RELATIONS TOTALES/PARTIELLES*

- Pour les dépendances fonctionnelles inter-relations (ou plus exactement référentielles).
- Une dépendance fonctionnelle est totale si et seulement si toutes les valeurs du but potentiel sont effectivement référencées dans la relation source. On dit qu'il y a une contrainte de totalité.
- Une dépendance fonctionnelle est partielle si et seulement si certaines valeurs du but potentiel peuvent ne pas être atteintes par la df. C'est une situation par défaut.

# *EXEMPLE DE DÉPENDANCES FONCTIONNELLES TOTALES*

- Une dépendance fonctionnelle totale sera représentée par une flèche annotée par un T au niveau du but.
- Toute voiture a une puissance. Donc la totalité du couple (marque, modèle) se trouve référencée dans la relation Puissance. Par contre, la nationalité d'un constructeur de véhicules peut être dans la base sans pour autant que l'on dispose d'un véhicule de ce constructeur.

Donc la df inter-relation suivante est totale :

Voiture.Numéro d'immatriculation  $\longrightarrow$  (T) Puissance.(Marque,Modèle)  
 $\Rightarrow$

Voiture.Numéro d'immatriculation  $\longrightarrow$  (T) Puissance.Marque

et Voiture.Numéro d'immatriculation  $\longrightarrow$  (T) Puissance.Modèle

# *PREMIÈRE EXTENSION*

VOITURE			
Numéro d'immatriculation	Marque	Modèle	Couleur
91253 ABC 75	Renault	Twingo S	Bleu
56465 CDE 78	FIAT	Panda AM 96	Rouge
3456 FGR 75	GMC	D 350	Bleu
132 YZ 01	Mercedes	S 280	Gris métallisé
1256 DCE 75	GMC	S 280	Azur
8953 XY 56	Mercedes	D 250	Rouge

Voiture.Numéro d'immatriculation → (T) Puissance.(Marque,Modèle)

Voiture.Numéro d'immatriculation

→ (T) Puissance.Marque

Voiture.Numéro d'immatriculation

→ (T) Puissance.Modèle

PUISSANCE		
Marque	Modèle	Puissance
Renault	Twingo S	6 CV
FIAT	Panda AM 96	5 CV
GMC	D 350	12 CV
Mercedes	S 280	12 CV
GMC	S 280	8 CV
Mercedes	D 250	10 CV

## DEUXIÈME EXTENSION

VOITURE			
Numéro d'immatriculation	Marque	Modèle	Couleur
91253 ABC 75	Renault	Twingo S	Bleu
56465 CDE 78	FIAT	Panda AM 96	Rouge
3456 FGR 75	GMC	D 350	Bleu
132 YZ 01	Mercedes	S 280	Gris métallisé
8953 XY 56	Mercedes	D 250	Rouge

Voiture.Numéro d'immatriculation  $\longrightarrow$  ~~(T)~~ Puissance.(Marque,Modèle)

Par contre,

Voiture.Numéro d'immatriculation

$\longrightarrow$  (T) Puissance.Marque

Voiture.Numéro d'immatriculation

$\longrightarrow$  (T) Puissance.Modèle

PUISSANCE		
Marque	Modèle	Puissance
Renault	Twingo S	6 CV
FIAT	Panda AM 96	5 CV
GMC	D 350	12 CV
Mercedes	S 280	12 CV
GMC	S 280	8 CV
Mercedes	D 250	10 CV

# *CONTRAÎNTE DE DÉPENDANCES FONCTIONNELLES TOTALES*

Voiture.Numéro d'immatriculation  $\longrightarrow$  (T) Puissance.(Marque,Modèle)

$\text{PUISSANCE}[\text{marque}, \text{modele}] = \text{VOITURE}[\text{marque}, \text{modele}] \quad \Leftrightarrow$

$\text{VOITURE}[\text{marque}, \text{modele}] \subseteq \text{PUISSANCE}[\text{marque}, \text{modele}] \quad : \textbf{FK}$

et  $\text{PUISSANCE}[\text{marque}, \text{modele}] \subseteq \text{VOITURE}[\text{marque}, \text{modele}]$  donc il faut :  
 $\text{PUISSANCE}[\text{marque}, \text{modele}] - \text{VOITURE}[\text{marque}, \text{modele}] = \emptyset$

$\Rightarrow$

Voiture. Numéro d'immatriculation  $\longrightarrow$  (T) Puissance.Marque

$\text{PUISSANCE}[\text{marque}] = \text{VOITURE}[\text{marque}] \quad \Leftrightarrow$

$\text{VOITURE}[\text{marque}] \subseteq \text{PUISSANCE}[\text{marque}] \quad : \textbf{FK}$

et  $\text{PUISSANCE}[\text{marque}] \subseteq \text{VOITURE}[\text{marque}]$

et Voiture. Numéro d'immatriculation  $\longrightarrow$  (T) Puissance.Modèle

$\text{PUISSANCE}[\text{modele}] = \text{VOITURE}[\text{modele}] \quad \Leftrightarrow$

$\text{VOITURE}[\text{modele}] \subseteq \text{PUISSANCE}[\text{modele}] \quad : \textbf{FK}$

et  $\text{PUISSANCE}[\text{modele}] \subseteq \text{VOITURE}[\text{modele}]$

# *CONTRAÎNTE DE DÉPENDANCES*

## *FONCTIONNELLES INTER-RELATIONS TOTALES*

Malheureusement, ces contraintes sont difficilement implantables dans les SGBDR.

Elles devraient conduire à des insertions simultanées dans plusieurs relations. Elles peuvent être implantées au niveau des applications.

Oracle permet de programmer des procédures stockées au niveau du SGBDR à l'aide du langage PL/SQL. Une procédure stockée d'insertion peut exiger et gérer l'insertion simultanée dans plusieurs tables. Cette procédure peut être accessible à toutes les applications qui souhaitent faire des insertions. Il suffit alors de leur retirer la possibilité de la court-circuiter en leur retirant le privilège INSERT sur les tables.

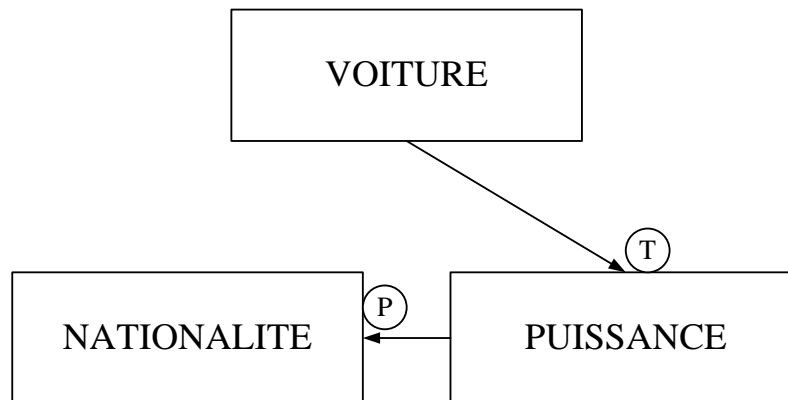
Oracle permet de gérer aussi des destructions simultanées grâce aux contraintes ON DELETE CASCADE.

# *ORDRE DE CRÉATION DES TABLES*

Les dépendances fonctionnelles inter-relations impliquent un ordre de création de tables (sauf en cas de références circulaires).

Un graphe des dépendances fonctionnelles inter-relation permet de déterminer l'ordre de création des tables (sauf en cas de références circulaires).

Ainsi suivant le graphe suivant, il faudrait créer dans l'ordre les tables :



- (1) NATIONALITE
- (2) PUISSANCE
- (3) VOITURE



# *DD DE LA BASE LOCATION DE VOITURE*

<b>CODE</b>	<b>LIBELLE</b>	<b>NATURE</b>	<b>CATEGORIE</b>	<b>DOMAINE</b>	<b>COMMENTAIRE</b>
NUM_IMMA	Numéro d'immatriculation	Non calculée / saisie	Stable	texte[20] obligatoire	Identifiant de voiture
MARQUE	Marque	Non calculée / saisie	Stable	Texte[30] obligatoire	
MODELE	Modèle	Non calculée / saisie	Stable	Texte[30] obligatoire	
COULEUR	Couleur du véhicule	Non calculée / saisie	Evolutive	Texte[10] dans ( « Bleu », « Rouge », « Vert », « Noire »)	
NAT_CONS	Nationalité du constructeur	Non calculée / saisie	Stable	texte[20] obligatoire	
PUISSANCE	Puissance du moteur	Non calculée / saisie	Stable	Numérique Obligatoire	

# *SCHEMA LOGIQUE DE LOCATION DE VOITURE*

Nationalité (Marque, Nat\_ Cons)

Puissance (#Marque, Modele, Puissance)

Voiture (Num\_imma, #[Marque, Modele], Couleur)

*Ce schéma logique a été déduit de :*

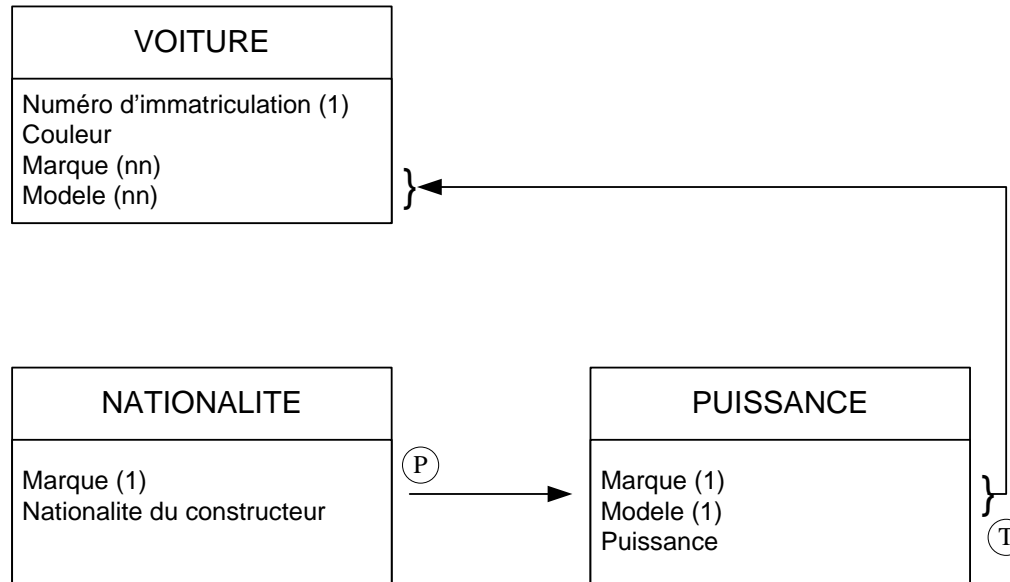
$$F = \{ \begin{array}{l} \text{Num\_imma} \rightarrow \text{Marque}, \\ \text{Num\_imma} \rightarrow \text{Modele}, \\ \text{Num\_imma} \rightarrow \text{Couleur}, \\ \text{Marque} \rightarrow \text{Nat\_Cons}, \\ \text{Marque, Modele} \rightarrow \text{Puissance} \end{array} \}$$

*Cependant, un tel schéma n'exprime pas toutes les contraintes connues :*

`PUISSANCE[marque, modele]  $\subseteq$  VOITURE[marque, modele]`

`VOITURE.[#marque, modele] NOT NULL`

# *SCHEMA RELATIONNEL*



## Schéma relationnel - Légende :

- (1) : partie de la clé primaire donc (nn,uq)
- { } : clé composée
- (nn) : clé étrangère obligatoire
- (uq) : clé étrangère avec contrainte d'unicité
- (P) : df inter-relation partielle
- (T) : df inter-relation totale
- ← : référence

# *LA NORMALISATION D'UN SCHÉMA RELATIONNEL*

Le modèle relationnel offre la notion de "bon schéma " et une approche de transformation d'un schéma en "un bon schéma".

Cette approche s'appelle la NORMALISATION. Elle conduit à un schéma composé de schémas de relations dites en Formes Normales.

Au début, on part d'une table unique (relation universelle), à la fin on a plusieurs tables, toutes en 3ième FN (ou plus).

**Elle consiste à éliminer les redondances afin d'éviter les difficultés de maintien de cohérence en insertion, en mise à jour et en suppression. De plus elle permet une meilleure représentation du réel.**

**NORMALISER = DECOMPOSER SANS PERTE D'INFORMATION (pour les premières formes normales).**

# *NORMALISATION À PARTIR DE LA RELATION UNIVERSELLE*

- Cette méthode consiste à créer une table universelle contenant toutes les données à stocker du dictionnaire des données.
- A cette table, on va donner une clé, généralement l'identifiant de l'objet central ou fédérateur de la base de données.
- Ensuite on décompose cette table en appliquant successivement les trois premières formes normales.

# *NORMALISATION À PARTIR DE LA RELATION UNIVERSELLE*

- Toutes les tables issues de la décomposition devront être conservées à l'exception des tables uniquement formées d'une clé primaire simple (les tables constituées d'une seule clé primaire composée doivent absolument être conservées).
- Le besoin en données identifiantes, qui correspond aux manques du graphe des dépendances fonctionnelles, apparaîtra lorsque l'on devra donner une clé primaire aux nouvelles tables issues de la décomposition. Il suffira alors de créer *ex-nihilo* une donnée identifiante de type incrémentée/compteur.

# *LA PREMIÈRE FORME NORMALE*

- Une relation est en première forme normale si tout attribut contient une valeur atomique. Un attribut, donnée élémentaire du monde réel, ne peut désigner, ni une donnée composée d'entités de nature quelconque, ni une liste de données de même nature.
  - Les groupes répétitifs, c'est à dire les attributs à occurrence multiple, correspondant non pas à des valeurs atomiques mais à des listes de valeurs, sont interdits.
  - Un groupe répétitif est constitué d'un attribut ou d'un groupe d'attributs pour lesquels il existe plusieurs valeurs pour une même valeur de la clé primaire.
- 1FN : Valeur atomique + la clé détermine les autres att...**

# LA PREMIÈRE FORME NORMALE

Table EMPLOYES							
Numéro Employé	Nom	Prénom	S.G.B.D. maîtrisés	Intérêts	Nature diplôme	Lieu du diplôme	Date du diplôme
01	DURAND	Jacques	Oracle	Dessin	BAC	Versailles	1990
			Access	Peinture			
			SQL Server	Théâtre	D.E.A.	Paris	1996
			Ingres				
02	DUPONT	Jeanne	DBase	Volley-ball	BAC	Aix	1989
			Paradox	Randonnée	Ingénieur	Marseille	1994
					Docteur	Marseille	1997

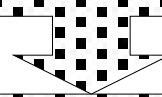


Table EMPLOYES			Table S.G.B.D.		Table INTERETS DES EMPLOYES	
Numéro Employé	Nom	Prénom	Numéro employé	SGBD maîtrisés	Numéro employé	Intérêts
01	DURAND	Jacques	01	Oracle	01	Dessin
02	DUPONT	Jeanne	01	Access	01	Peinture
			01	SQL Server	01	Théâtre
			01	Ingres	02	Volley-ball
			02	DBase	02	Randonnée
			02	Paradox		

Table DIPLOMES DES EMPLOYES				
Numéro diplôme	Numéro employé	Nature diplôme	Lieu du diplôme	Date du diplôme
01	01	BAC	Versailles	1990
02	01	D.E.A.	Paris	1996
03	02	BAC	Aix	1989
04	02	Ingénieur	Marseille	1994
05	02	Docteur	Marseille	1997

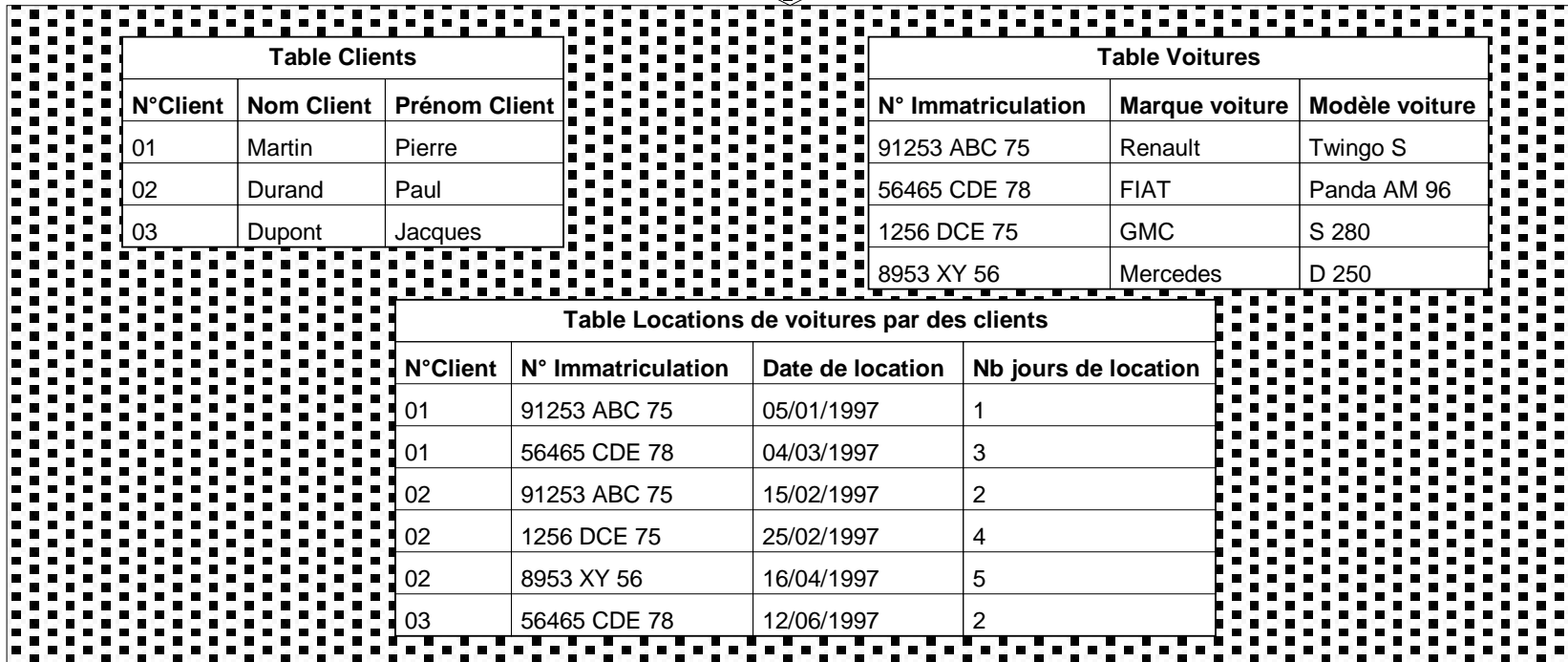


# *LA DEUXIÈME FORME NORMALE*

- Le passage en deuxième forme normale ne concerne que les tables ayant une clé primaire composée et en première forme normale. Une table en première forme normale disposant d'une clé primaire simple est automatiquement en deuxième forme normale.
- La règle de normalisation exige que tout attribut ne faisant pas partie de la clé primaire dépende de la totalité de la clé primaire (DF élémentaire). Si on trouve un attribut qui ne dépend que d'une partie de la clé primaire, il doit être exclu. Ainsi, on interdit une table décrivant plusieurs types d'objets distincts.
- Pour chaque groupe d'attributs ne dépendant que d'une partie de la clé primaire :
  - Rassembler dans une autre table les attributs ne dépendant que d'une partie de la clé primaire de la table initiale.
  - Donner cette partie de clé primaire initiale comme clé primaire de la nouvelle table.

# LA DEUXIÈME FORME NORMALE

Table Location de voitures par des clients							
N°Client	Nom Client	Prénom Client	N° Immatriculation	Marque voiture	Modèle voiture	Date de location	Nb jours de location
01	Martin	Pierre	91253 ABC 75	Renault	Twingo S	05/01/1997	1
01	Martin	Pierre	56465 CDE 78	FIAT	Panda AM 96	04/03/1997	3
02	Durand	Paul	91253 ABC 75	Renault	Twingo S	15/02/1997	2
02	Durand	Paul	1256 DCE 75	GMC	S 280	25/02/1997	4
02	Durand	Paul	8953 XY 56	Mercedes	D 250	16/04/1997	5
03	Dupont	Jacques	56465 CDE 78	FIAT	Panda AM 96	12/06/1997	2



# *LA TROISIÈME FORME NORMALE*

- La troisième forme normale a pour but d'éliminer les dépendances fonctionnelles transitives au sein d'une table. La table doit au préalable être en deuxième forme normale.
- Ainsi, au sein d'une table, un attribut ne faisant pas partie de la clé doit dépendre directement de la clé, et d'aucun autre attribut ou groupe d'attributs.
- Le passage en troisième forme normale impose alors de :
  - Mettre dans une autre table les attributs qui dépendent fonctionnellement d'un attribut ou d'un groupe d'attributs autres que la clé.
  - Faire apparaître dans les deux tables ce dernier attribut ou groupe d'attributs, qui servira de clé primaire à la nouvelle table.

# LA TROISIÈME FORME NORMALE

Table Voitures

Numéro d'immatriculation	Marque	Modèle	Puissance	Nationalité du constructeur	Couleur
91253 ABC 75	Renault	Twingo S	6 CV	Française	Bleu
56465 CDE 78	FIAT	Panda AM 96	5 CV	Italienne	Rouge
132 YZ 01	Mercedes	S 280	12 CV	Allemande	Gris métallisé
1256 DCE 75	GMC	S 280	8 CV	Américaine	Azur
8953 XY 56	Mercedes	D 250	10 CV	Allemande	Rouge
4565 TD 34	FIAT	Punto	6 CV	Italienne	Noir
7897 BDE 75	Renault	Twingo S	6 CV	Française	Beige
2536 BRF 91	Mercedes	S 280	12 CV	Allemande	Bleu

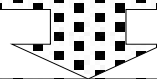


Table voitures

Numéro d'immatriculation	Marque	Modèle	Couleur
91253 ABC 75	Renault	Twingo S	Bleu
56465 CDE 78	FIAT	Panda AM 96	Rouge
132 YZ 01	Mercedes	S 280	Gris métallisé
1256 DCE 75	GMC	S 280	Azur
8953 XY 56	Mercedes	D 250	Rouge
4565 TD 34	FIAT	Punto	Noir
7897 BDE 75	Renault	Twingo S	Beige
2536 BRF 91	Mercedes	S 280	Bleu

Table Nationalités par marque

Marque	Nationalité du constructeur
Renault	Française
FIAT	Italienne
Mercedes	Allemande
GMC	Américaine

Table Puissances par modèle

Marque	Modèle	Puissance
Renault	Twingo S	6 CV
FIAT	Panda AM 96	5 CV
Mercedes	S 280	12 CV
GMC	S 280	8 CV
Mercedes	D 250	10 CV
FIAT	Punto	6 CV



# *LA NORMALISATION D'UN SCHÉMA RELATIONNEL*

L'ensemble des DF sous-jacent à une collection de relations 3FN est une "couverture minimale". Il n'y a, dans cet ensemble, aucune DF inutile (ce qui n'est pas le cas d'un graphe autre).

L'interprétation concrète de la minimalité est la suivante : la normalisation conduit à l'émergence d'objets (les relations) qui minimisent l'expression de leurs relations structurelles.

**La normalisation jusqu'en 3<sup>ième</sup> forme normale:**

- **Élimine en grande partie le problème de redondance.**
- **Décompose la relation universelle en plusieurs tables sans perte d'information.**
- **permet une meilleure connaissance du réel mais l'éclate dans plusieurs tables ce qui peut nuire à l'indépendance logique.**



# Chapitre 9

## Rappels

# Modèle physique (Oracle SQL2)

Michel Dubois

I.U.T. de Vannes

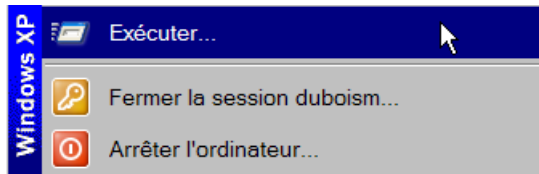
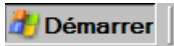
*Michel.Dubois@univ-ubs.fr*

[Retour au plan](#)

# SQLPLUS

- Invocation du client SQLPLUS (sur les partitions debian/linux à l'IUT, la commande `sqlplus10g` est aussi disponible)
  - `>sqlplus /`  
Si les comptes ont une authentification externe à Oracle.
  - `>sqlplus duboism/xxx@oraetud`  
Si le nom de service `oraetud` a été définie dans `tnsnames.ora`.
  - `>sqlplus duboism/xxx@"(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=oraetud.univ-ubs.fr)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=oraetud)))"`  
Si aucun nom de service n'a été défini, on peut toujours utiliser une chaîne de connexion *service Oracle net*. Il faut cependant que dans le fichier `oracle/product\10.2.0\client_3\NETWORK\ADMIN\sqlnet.ora` il y ait : `SQLNET.AUTHENTICATION_SERVICES= (RADIUS)`
  - `>sqlplus`  
Entrez le nom utilisateur : `duboism@oraetud`  
Entrez le mot de passe : `xxx`  
Si on ne précise rien, il y a une demande interactive.
  - `>sqlplus /nologin`  
`SQL>connect duboism/xxx@oraetud;`  
La commande `SQLPLUS connect` est toujours disponible.
- L'invocation d'un script sql est possible :
  - `sqlplus duboism/xxx@oraetud @script.sql parametre1 parametre2 ...`
  - Les variables `&1` et `&2` dans le script SQL `script.sql` prendront les valeurs `parametre1` et `parametre2`.
  - Le script SQL doit posséder en fin la commande `EXIT`.
  - Les commandes `SQLPLUS start script.sql` ou `@script.sql` ou `@script` permettent d'exécuter le fichier de commande `script.sql` du répertoire courant (celui d'où a été invoqué `sqlplus`).
  - Sous windows XP, il vaut mieux utiliser l'invite de commandes que la version Windows de `SQLPLUS`
- La commande `SQLPLUS EXIT` permet de quitter `SQLPLUS`.
- Les commandes `SQLPLUS !` ou `HOST` permettent d'exécuter une commande du système d'exploitation
  - Sous windows XP, la commande `host` est disponible mais pas `!`.
  - `!cp -p /forum/vannes/prof/lcsd01/MD/xxx.sql ~` (copier un fichier)
  - `!cd /forum/vannes/prof/lcsd01/MD` (changer le répertoire courant : ne marche que pour Linux)
  - `host sqlldr10g userid=duboism/xxx@oraetud control=xxx.ctl` (charger une table sous linux à l'IUT)
  - `host \monRepertoire\xxx.bat` ou `!/monRepertoire/xxx.sh` (exécuter un fichier de commandes externes)
- Les script shell `*.sh` sous linux et les fichiers `*.bat` sous Windows XP permettent de l'invoquer en mode batch.

# SQLPLUS MSDOS



```
C:\WINDOWS\System32\CMD.exe - sqlplus /
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\duboism>cd \

C:\>cd \boaccess\ora_olap

C:\boaccess\ora_olap>sqlplus /

SQL*Plus: Release 8.1.7.0.0 - Production on Ma Jan 3 16:48:26 2006

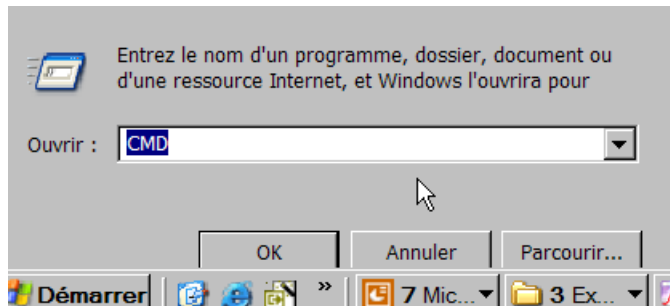
(c) Copyright 2000 Oracle Corporation. All rights reserved.

ERROR:
ORA-01017: invalid username/password; logon denied

Entrez le nom utilisateur : ops$duboism
Entrez le mot de passe :

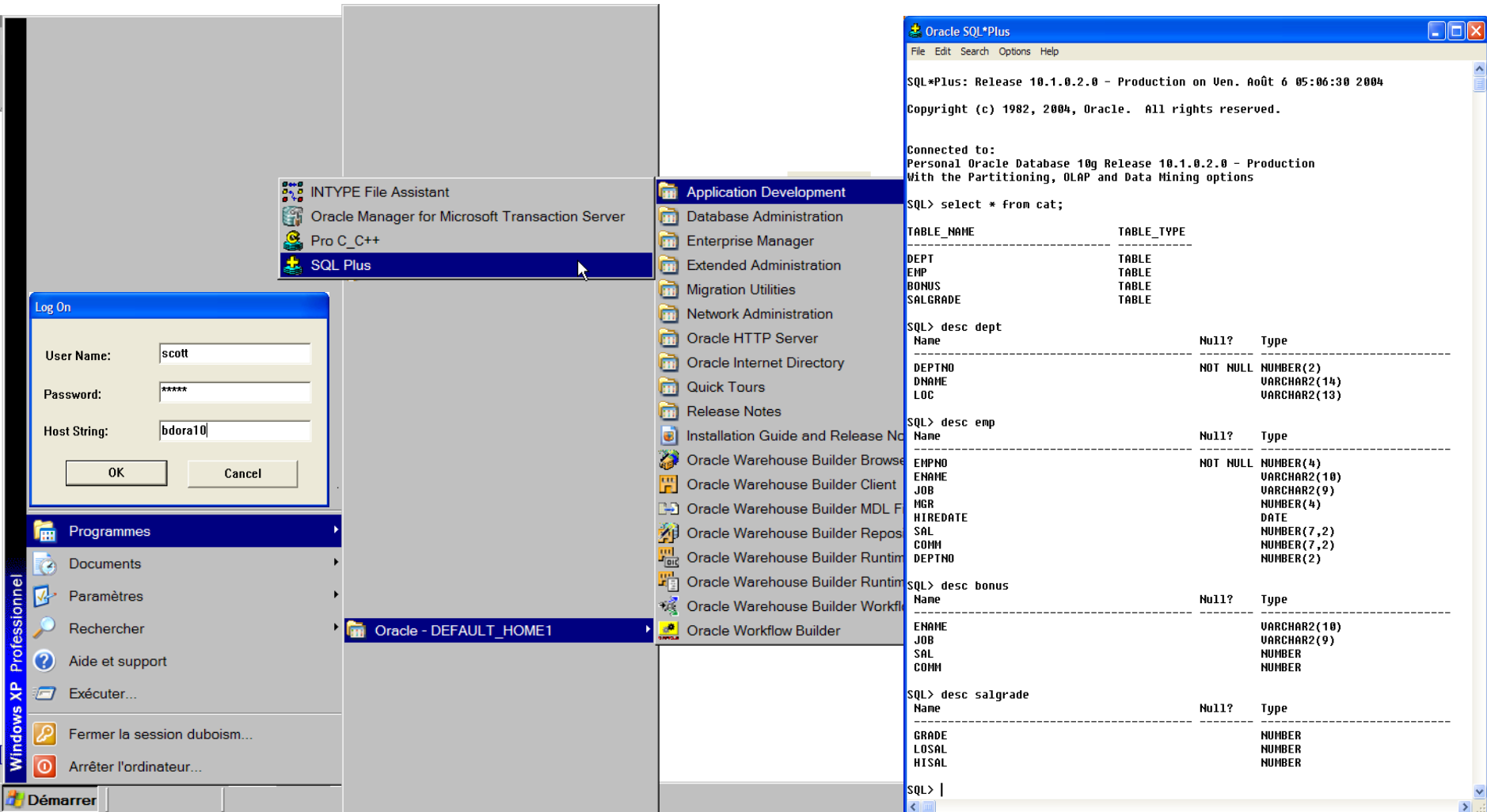
Connect0 α :
Personal Oracle8i Release 8.1.7.0.0 - Production
With the Partitioning option
JServer Release 8.1.7.0.0 - Production

SQL> _
```





# SQLPLUS WINDOWS



# *SQLPLUS/SQLldr LINUX*

`sqlplus dubois/xxxxxx@oraetud` (à partir des distributions debian locales comme pour Windows)

`sqlplus10g dubois/xxxxxx@oraetud` (à partir des distributions debian locales)

`sqlldr10g userid=dubois/xxxxxx@oraetud  
control=xxx.ctl`

# *LES CONTRAINTES D'INTÉGRITÉS*

Le relationnel possède, outre la normalisation, 3 contraintes de modèle:

Un attributs à un domaine. Toute valeur insérée dans un attribut doit être compatible avec son domaine.

Une table à une clé. La valeur de cette clé doit être unique.

L'intégrité référentielle vise à vérifier que la clé étrangère référence une clé primaire. Toute valeur (autre que NULL) qui est insérée dans la clé étrangère doit au préalable exister dans la clé primaire.

D'autres contraintes peuvent être ajoutées (en fonction de l'analyse):

Des attributs peuvent être multiples ou uniques.

Des attributs peuvent être facultatifs ou obligatoires.

Des contraintes liées au domaine peuvent être implémentées soit de façon déclarative, soit à l'aide de déclencheurs.

# *CONTRAİNTE SUR UNE COLONNE*

- Les règles portant sur une colonne seulement sont déclarées immédiatement après le type de la colonne.
- `Null/Not Null` : interdit que la colonne puisse prendre des valeurs nulles (non-définies).
- `Unique` : la colonne est une clé (candidate) de la table. Les valeurs de cette colonne sont toutes distinctes les unes des autres. (La colonne doit aussi être spécifiée comme not null)
- `Primary Key` : identique à unique. La clé primaire est la clé que le concepteur privilégie dans les accès et les jointures avec cette table, elle est indexée.
- `References . . .` : les valeurs de cette colonne doivent être des éléments de la table projetée sur la colonne de référence. La colonne de référence doit être spécifiée comme unique (ou clé primaire)
- `Check ( . . . )` : un prédicat est associé à la colonne, il utilise seulement les noms des constituants de la table et se vérifie localement sur la rangée (ceci exclut les sous requêtes)

# *CONSTRAINTES SUR PLUSIEURS COLONNES: IE DE TUPLE*

- `Unique( ... )` : le groupe de colonnes est une clé de la table. Les valeurs de ces colonnes sont toutes distinctes les unes des autres. (Les colonnes doivent aussi être spécifiées comme not null)
- `Primary Key( ... )` : identique à la clause unique. Une seule clé primaire est autorisée par table.
- `Foreign Key ( ) References ...` : les valeurs de ce groupe de colonnes doivent être des éléments de la table projetée sur le groupe de référence. Les colonnes de référence doivent être spécifiées comme unique (ou clé primaire)
- `Check( ... )` : déjà vu plus haut

# *CONSTRAINTES ET ORACLE*

Contraintes sur la relation	Implantation
Clé primaire	PRIMARY KEY
Clé(s) candidate(s)	UNIQUE
Clé(s) secondaire(s)	CREATE INDEX
Clé(s) étrangère(s) et provenance	FOREIGN KEY REFERENCES
Contrainte sur attribut autre que les clés	NULL vs NOT NULL, CHECK
Contrainte sur tuple appelée par oracle contrainte de table	CHECK
Contrainte sur table	TRIGGER
Contrainte d'ajout	DEFAULT, TRIGGER
Contrainte de modification	TRIGGER
Contrainte de suppression Michel Dubois	ON DELETE CASCADE, TRIGGER

## *CREATION D 'UNE TABLE*

La commande de création de table la plus simple ne comportera que le nom et le type de chaque colonne de la table.

On peut créer une table par la commande CREATE TABLE en spécifiant le nom et le type de chaque colonne.

A la création, la table sera vide mais un certain espace lui sera alloué :

```
CREATE TABLE nom_table
```

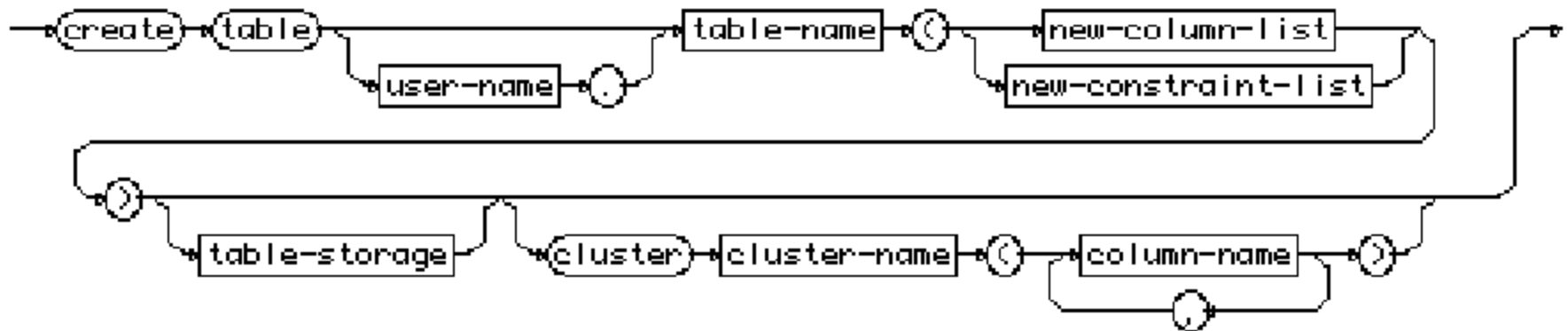
```
(nom_col1 TYPE1,
```

```
nom_col2 TYPE2,
```

```
...)
```

# *CREATE TABLE*

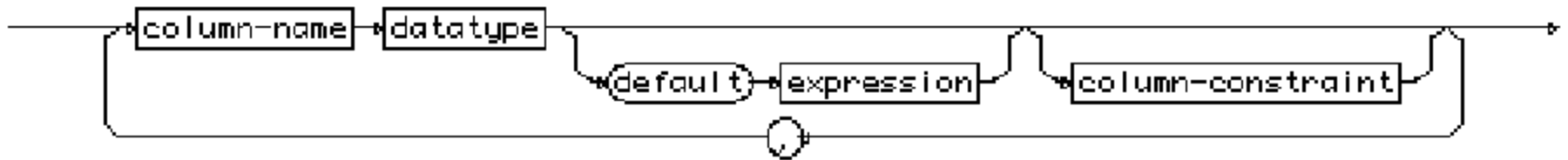
## CREATE-TABLE





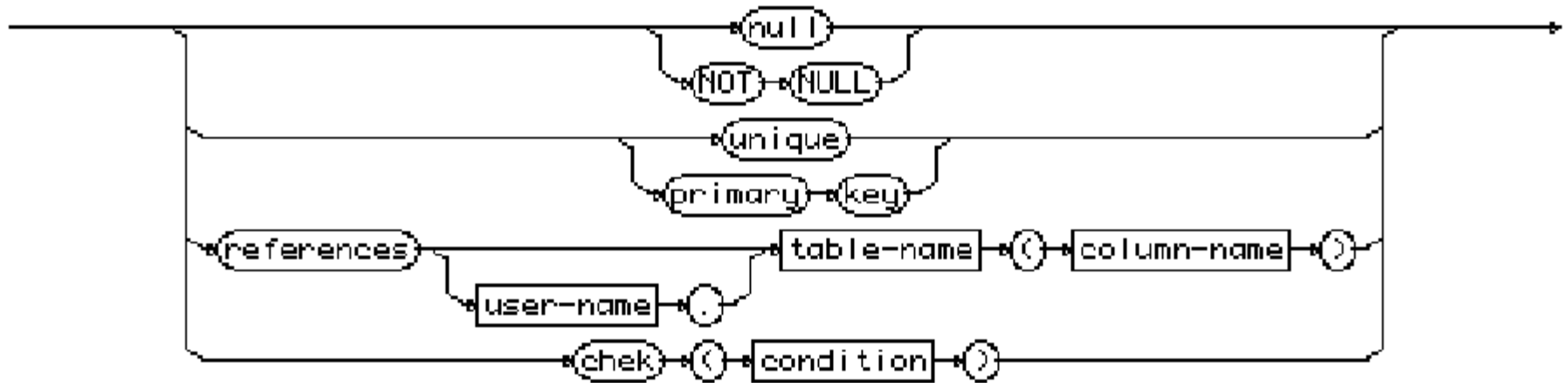
# *LISTE DES COLONNES*

new-column-list



# CONTRAINTE DE COLONNE

column-constraint



# *CONTRAİNTE DE TUPLE/TABLE*

new-constraint-list

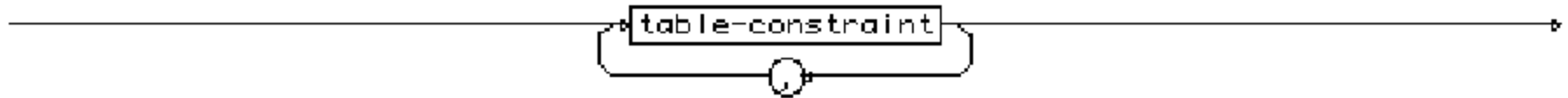
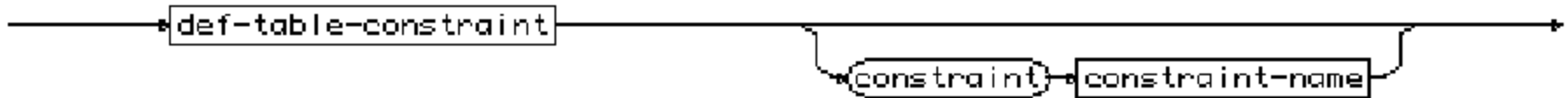
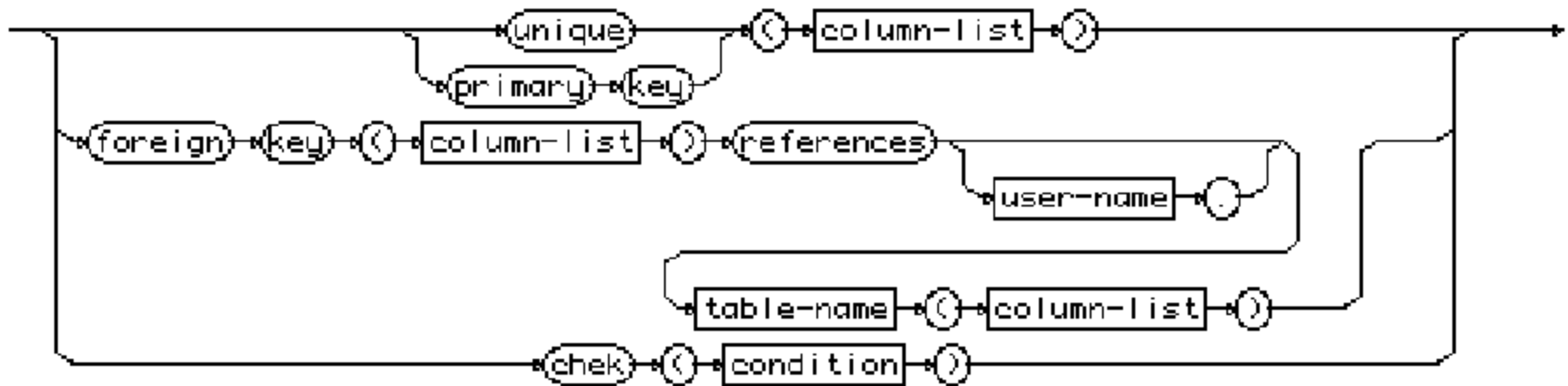


table-constraint



# *DÉFINITION DE CONTRAINTES DE TUPLE*

def-table-constraint



# *INSERTION & CREATION*

```
CREATE TABLE nom_table  
[ (nom_col1, nom_col2, ... ) ]  
AS SELECT...
```

On n'a pas besoin alors de spécifier de type pour les colonnes : les types des données sont ceux provenant du SELECT.

Si des conversions de type sont à faire, on peut dans le SELECT utiliser les fonctions TO CHAR, TO DATE, TO NUMBER.

Par défaut les noms des colonnes de la nouvelle table sont les noms des colonnes du SELECT. Si des expressions apparaissent dans le SELECT, les colonnes correspondantes doivent impérativement être renommées.

Le SELECT peut contenir des fonctions de groupes mais pas de ORDER BY car les lignes d'une table ne peuvent pas être classées.

Mais il reste à définir les contraintes d'intégrité que devront respecter les données que l'on mettra dans la table. A noter qu'il faut que les tuples déjà insérés respectent ces contraintes qui vont être rajoutée après.

# *MODIFIER UNE TABLE*

- On peut modifier dynamiquement la définition d'une table grâce à la commande `ALTER TABLE`.
- Deux types de modifications sont possibles : ajout d'une colonne et modification d'une colonne existante.
- Il n'est pas possible de supprimer une colonne. Par contre une colonne qui n'est plus utilisée peut être mise à la valeur `NULL`, auquel cas elle n'occupe plus d'espace disque. Si on désire vraiment supprimer une colonne, il faut :
  - se créer une nouvelle table sans la colonne en question
  - détruire l'ancienne table, après insertion des valeurs de l'ancienne,
  - donner à la nouvelle table le nom de l'ancienne.

# *AJOUTER UNE COLONNE*

- La commande suivante permet d'ajouter une ou plusieurs colonnes à une table existante :

```
ALTER TABLE nom_table ADD  
(nom_col1 TYPE1,  
 nom_col2 TYPE2,  
 ... )
```

- Les types possibles sont les mêmes que ceux décrits avec la commande CREATE TABLE.
- Si la table contient déjà des lignes, la nouvelle colonne aura des valeurs NULL pour les lignes existantes.

# *MODIFIER UNE COLONNE*

```
ALTER TABLE table
```

```
MODIFY (col1 type1, col2 type2, ...)
```

Il est possible de modifier la définition d'une colonne, à condition que la nouvelle définition soit compatible avec le contenu de la colonne et en respectant les contraintes suivantes :

- dans tous les cas il est possible d'augmenter la taille d'une colonne;
- il est possible de diminuer la taille, ou même de changer le type d'une colonne vide ;
- on peut spécifier NOT NULL si la colonne ne contient aucune valeur NULL ;
- on peut dans tous les cas spécifier NULL pour autoriser les valeurs NULL.



# *SUPPRIMER UNE TABLE*

- La commande DROP TABLE permet de supprimer une table :

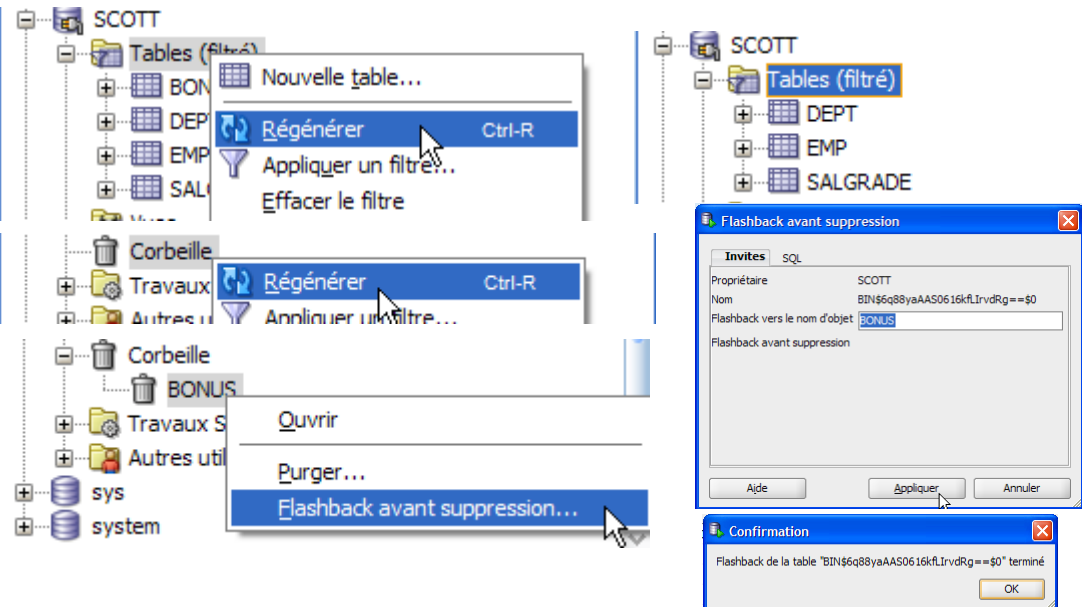
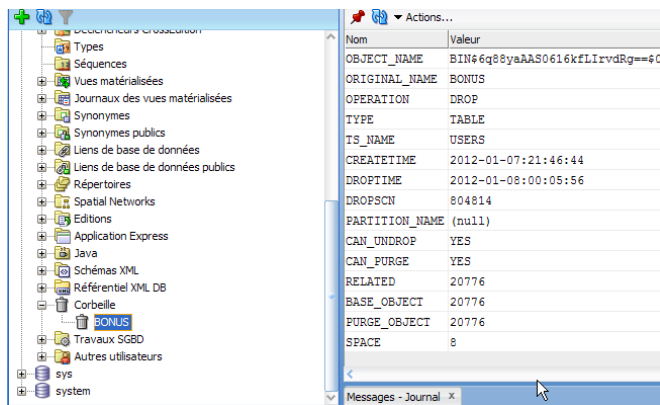
```
DROP TABLE utilisateur.nom_table  
[CASCADE CONSTRAINTS] ;
```

- La table nom table est alors supprimée.
- La définition de la table ainsi que son contenu sont détruits, et l'espace occupé par la table est libéré.
- L'option CASCADE CONSTRAINTS permet de supprimer toutes les contraintes d'intégrités référentielles qui font référence à la clé primaire de la table.

# *CORBEILLE POUR ORACLE 10g & ORACLE SQL DEVELOPER*

- Les commandes DROP XXX où XXX est un objet oracle comme une table, une vue, etc. ne sont pas définitives tant que l'on a pas fait un COMMIT.
- L'application ORACLE SQL DEVELOPER propose une gestion graphique de la corbeille : Pour la restaurer :

**DROP TABLE BONUS ;**



**FLASHBACK TABLE "SCOTT"."BIN\$6q88yaAAS0616kfLIrvdRg==\$0"  
TO BEFORE DROP RENAME TO BONUS ;**

Pour la purger avant le commit :

**PURGE TABLE "SCOTT"."BIN\$6q88yaAAS0616kfLIrvdRg==\$0" ;**

# *RENOMMER UNE TABLE*

- On a la possibilité de changer le nom d'une table par la commande RENAME.

la syntaxe est la suivante :

```
RENAME ancien_nom TO nouveau_nom ;
```

# *PROBLÈMES AVEC LES CONTRAINTES*

- Certaines contraintes peuvent empêcher l'ajout de nouvelles données
- Par exemple, si on ajoute une contrainte référentielle qui référence la table elle-même (cas de la hiérarchie employé/supérieur).
- Pour pallier ce problème, SQL 2 permet d'indiquer que la vérification d'une contrainte doit être différée à la fin de la transaction en cours, et pas immédiatement pour chaque commande SQL.

# *CONSTRAINTES DIFFÉRABLES*

CONSTRAINT nom définition-contrainte  
[NOT] DEFERRABLE  
[INITIALLY {DEFERRED | IMMEDIATE} ]

- La valeur par défaut est NOT DEFERRABLE
- Par défaut, si elle peut être différée, elle ne l'est pas si on ne fait rien ; on peut changer ça en précisant INITIALLY DEFERRED

# *DIFFÉRER UNE CONTRAINTE*

- Si des contraintes sont DEFERRABLE, pour les différer (pour le temps de la transaction en cours) on doit lancer la commande :

```
SET CONSTRAINTS {liste de  
contraintes | ALL} DEFERRED
```

- Si une contrainte est INITIALLY DEFERRED, on peut ne plus la différer (pour le temps de la transaction en cours) en lançant la commande :

```
SET CONSTRAINTS {liste de  
contraintes | ALL} IMMEDIATE
```

# *GESTION DES CONTRAINTES*

- Des contraintes d'intégrité peuvent être ajoutées ou supprimées par la commande ALTER TABLE. Mais pour modifier une contrainte, il faut la supprimer et ajouter ensuite la contrainte modifiée.

```
ALTER TABLE EMP
```

```
DROP CONSTRAINT NOM_UNIQUE
```

```
ADD (CONSTRAINT SAL_MIN CHECK(SAL + NVL(COMM,0) > 5000))
```

```
ALTER TABLE EMP
```

```
MODIFY NOME CONSTRAINT NOM_UNIQUE UNIQUE
```

- Il est parfois intéressant d'enlever temporairement des contraintes. Oracle le permet par la commande ALTER TABLE ... DISABLE/ENABLE.

```
ALTER TABLE table
```

```
DISABLE CONSTRAINT NOM_ UNIQUE UNIQUE
```

# ACTIVATION/REACTIVATION DE CONTRAINTES

- Si les contraintes ne sont pas désactivées, à chaque ajout d'une ligne le moteur de la base va devoir vérifier que cette ligne respecte les contraintes. De plus si les données ne sont pas chargées exactement dans le bon ordre, des contraintes de type intégrité référentielle peuvent être temporairement non validées.
- Il faut pouvoir aisément désactiver les contraintes en les ayant explicitement nommées lors de la création de la table.
- Oracle fournit un script (`utlexcpt.sql`) pour la création d'une table qui va servir à récupérer les éventuelles erreurs détectées suite à la réactivation des contraintes.
- Si les nouvelles données ne respectent pas les contraintes, ces dernières ne pourront être réactivées tant que les erreurs n'auront pas été corrigées.
- Si les contraintes ne peuvent être réactivées du fait que certaines données ne sont plus conformes, les enregistrements en cause seront référencés (par leur rowid) dans la table « exceptions » crée par le script « `utlexcpt.sql` ». Une fois les erreurs corrigées, l'opération de réactivation peut être renouvelée.

```
⇒ ALTER TABLE nom_table DISABLE CONSTRAINT nom_contrainte;
```

```
⇒ -- utlexcpt.sql
```

```
CREATE TABLE exceptions(row_id ROWID, owner VARCHAR2(30), table_name  
VARCHAR2(30), constraint VARCHAR2(30));
```

```
⇒ ALTER TABLE nom_table ENABLE CONSTRAINT nom_contrainte EXCEPTIONS INTO  
exceptions;
```

```
⇒ SELECT * FROM nom_table WHERE ROWID IN (SELECT row_id FROM exceptions)
```





# *VUE SUR LES COLONNES DES CONSTRAINTES*

- La vue USER\_CONS\_COLUMNS permet d'obtenir ces renseignements.

```
SQL> SELECT  constraint_name, column_name
      2  FROM    user_cons_columns
      3  WHERE   table_name = 'EMP';
```

CONSTRAINT_NAME	COLUMN_NAME
-----	-----
EMP_DEPTNO_FK	DEPTNO
EMP_EMPNO_PK	EMPNO
EMP_MGR_FK	MGR
SYS_C00674	EMPNO
SYS_C00675	DEPTNO



# *AJOUTER DES COMMENTAIRES A UNE TABLE*

```
SQL> COMMENT ON TABLE emp  
2 IS 'Employee Information';  
Comment created.
```

- Vous pouvez ajouter un commentaire à une table par l'instruction COMMENT.
- Les commentaires pourront être obtenus par les vues :
  - ALL\_COL\_COMMENTS
  - USER\_COL\_COMMENTS
  - ALL\_TAB\_COMMENTS
  - USER\_TAB\_COMMENTS



## *UNE SEQUENCE*

- C'est un générateur automatique de nombres uniques;
- C'est un objet partageable;
- C'est un objet utilisé pour créer des valeurs pour une clé primaire ;
- Cet objet permet d'éviter d'écrire du code au niveau applicatif ;
- Cet objet utilise le cache de la mémoire par soucis d'optimisation.

# CREATE / DROP SEQUENCE

```
CREATE SEQUENCE sequence
    [INCREMENT BY n]
    [START WITH n]
    [{MAXVALUE n | NOMAXVALUE}]
    [{MINVALUE n | NOMINVALUE}]
    [{CYCLE | NOCYCLE}]
    [{CACHE n | NOCACHE}] ;
```

```
SQL> DROP SEQUENCE dept_deptno;
Sequence dropped.
```



# *UTILISER UNE SEQUENCE*

Insertion d'un nouveau service  
“MARKETING” à San Diego.

```
SQL> INSERT INTO      dept(deptno, dname, loc)
      2  VALUES        (dept_deptno.NEXTVAL,
      3                  'MARKETING', 'SAN DIEGO');
1 row created.
```

Obtenir la valeur courante de la séquence  
DEPT\_DEPTNO.

```
SQL> SELECT    dept_deptno.CURRVAL
      2  FROM      dual;
```



# *VUE SUR LES SEQUENCES*

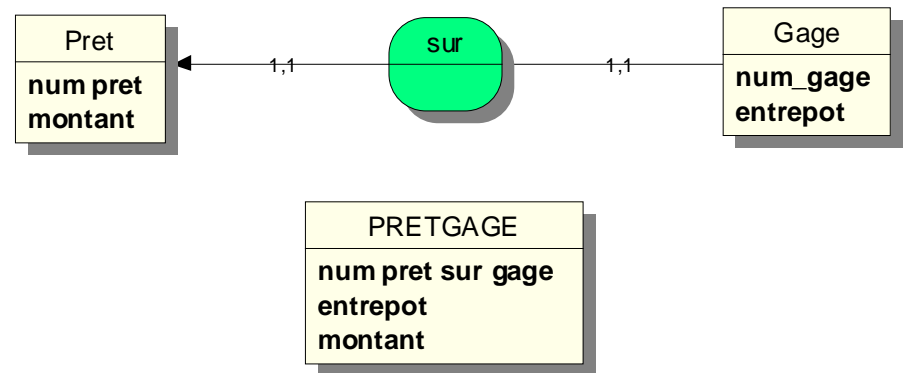
La vue USER\_SEQUENCES vous renseigne sur vos séquences.

```
SQL> SELECT  sequence_name, min_value, max_value,  
2           increment_by, last_number  
3 FROM      user_sequences;
```

Le prochain numéro de vos séquences se trouve dans la colonne LAST\_NUMBER.

# ASSOCIATION 1:1 -> (1,1)-(1,1)

- Une telle association peut se réduire à une seule entité.
- La règle n°1 s'applique. Des vues peuvent reconstituer les deux relations.



## *ASSOCIATION 1:1 -> (1,1)-(1,1)*

```
DROP TABLE PRETGAGE CASCADE CONSTRAINTS ;
```

PRETGAGE
<b>NUM_PRET_SUR_GAGE</b>
<b>ENTREPOT</b>
<b>MONTANT</b>

```
-- -----  
--          TABLE : PRETGAGE  
-- -----
```

```
CREATE TABLE PRETGAGE
```

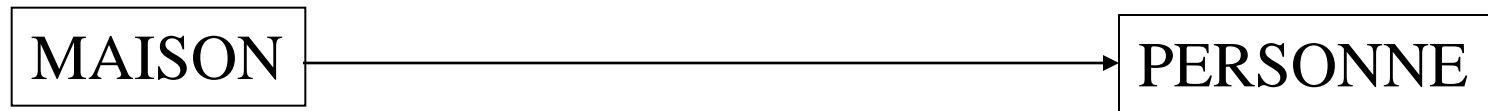
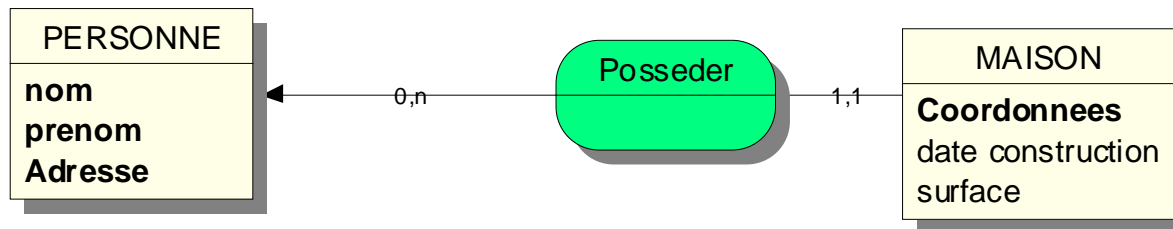
```
(  
  NUM_PRET_SUR_GAGE CHAR(32)  NOT NULL,  
  ENTREPOT CHAR(32)  NOT NULL,  
  MONTANT NUMBER(4)  NOT NULL,  
  CONSTRAINT PK_PRETGAGE PRIMARY KEY (NUM_PRET_SUR_GAGE)  
);
```

Michel Dubois



# ASSOCIATION 1:N -> (1,1)-(0,N)

- La règle n°2 s'applique.



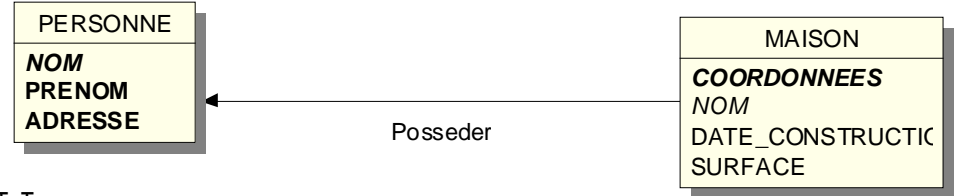
MAISON.#NOM est obligatoire

# *ASSOCIATION 1:N -> (1,1)-(0,N)*

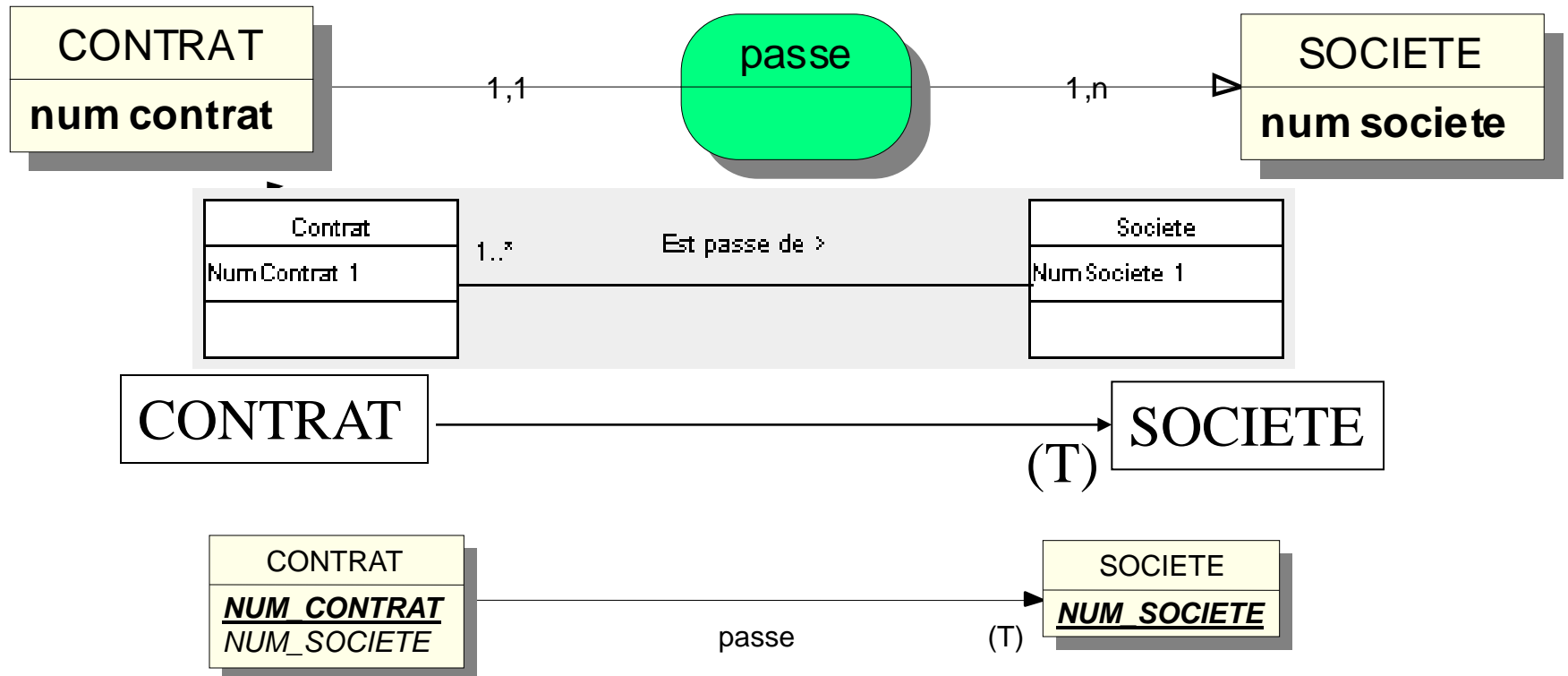
```
CREATE TABLE PERSONNE
(
  NOM CHAR(32) NOT NULL,
  PRENOM CHAR(32) NOT NULL,
  ADRESSE CHAR(32) NOT NULL,
  CONSTRAINT PK_PERSONNE PRIMARY KEY (NOM)
);
```

```
CREATE TABLE MAISON
```

```
(
  COORDONNEES CHAR(32) NOT NULL,
  NOM CHAR(32) NOT NULL,
  DATE_CONSTRUCTION DATE ,
  SURFACE NUMBER(4) ,
  CONSTRAINT PK_MAISON PRIMARY KEY (COORDONNEES) ,
  CONSTRAINT FK_POSSEDER FOREIGN KEY (NOM) REFERENCES PERSONNE (NOM)
);
```



# ASSOCIATION 1:N -> (1,1)-(1,N)



CONTRAT.num\_societe est obligatoire  
 SOCIETE[num\_societe]  $\subseteq$  CONTRAT[num\_societe]

- La règle n°2 s 'applique.

# *ASSOCIATION 1:N -> (1,1)-(1,N)*

SOCIETE[num\_societe]  $\subseteq$  CONTRAT[num\_societe]

```
CREATE TABLE CONTRAT
```

```
(
```

```
  NUM_CONTRAT CHAR(32) NOT NULL,
```

```
  NUM_SOCIETE CHAR(32) NOT NULL,
```

```
  CONSTRAINT PK_CONTRAT PRIMARY KEY (NUM_CONTRAT),
```

```
  CONSTRAINT FK_PASSE FOREIGN KEY (NUM_SOCIETE) REFERENCES  
  SOCIETE (NUM_SOCIETE)
```

```
);
```

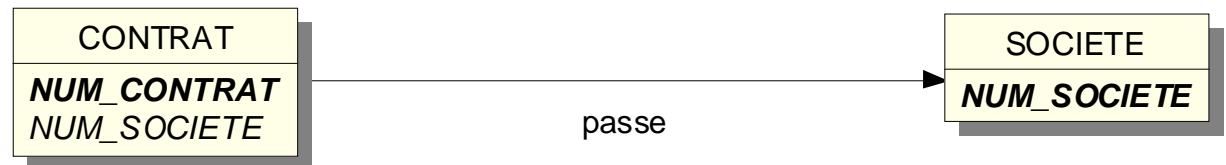
```
CREATE TABLE SOCIETE
```

```
(
```

```
  NUM_SOCIETE CHAR(32) NOT NULL,
```

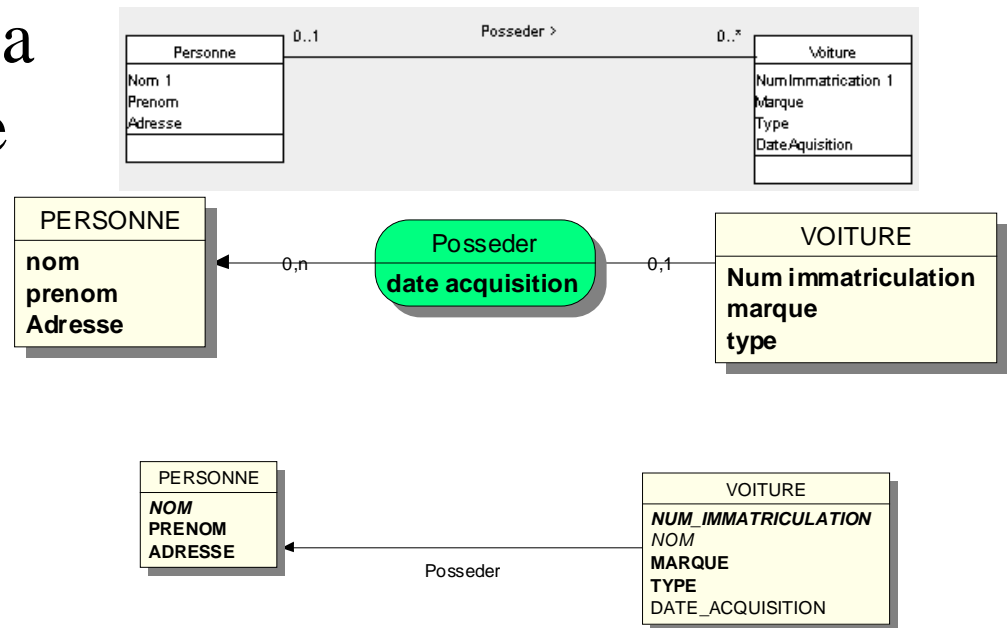
```
  CONSTRAINT PK_SOCIETE PRIMARY KEY (NUM_SOCIETE)
```

```
);
```

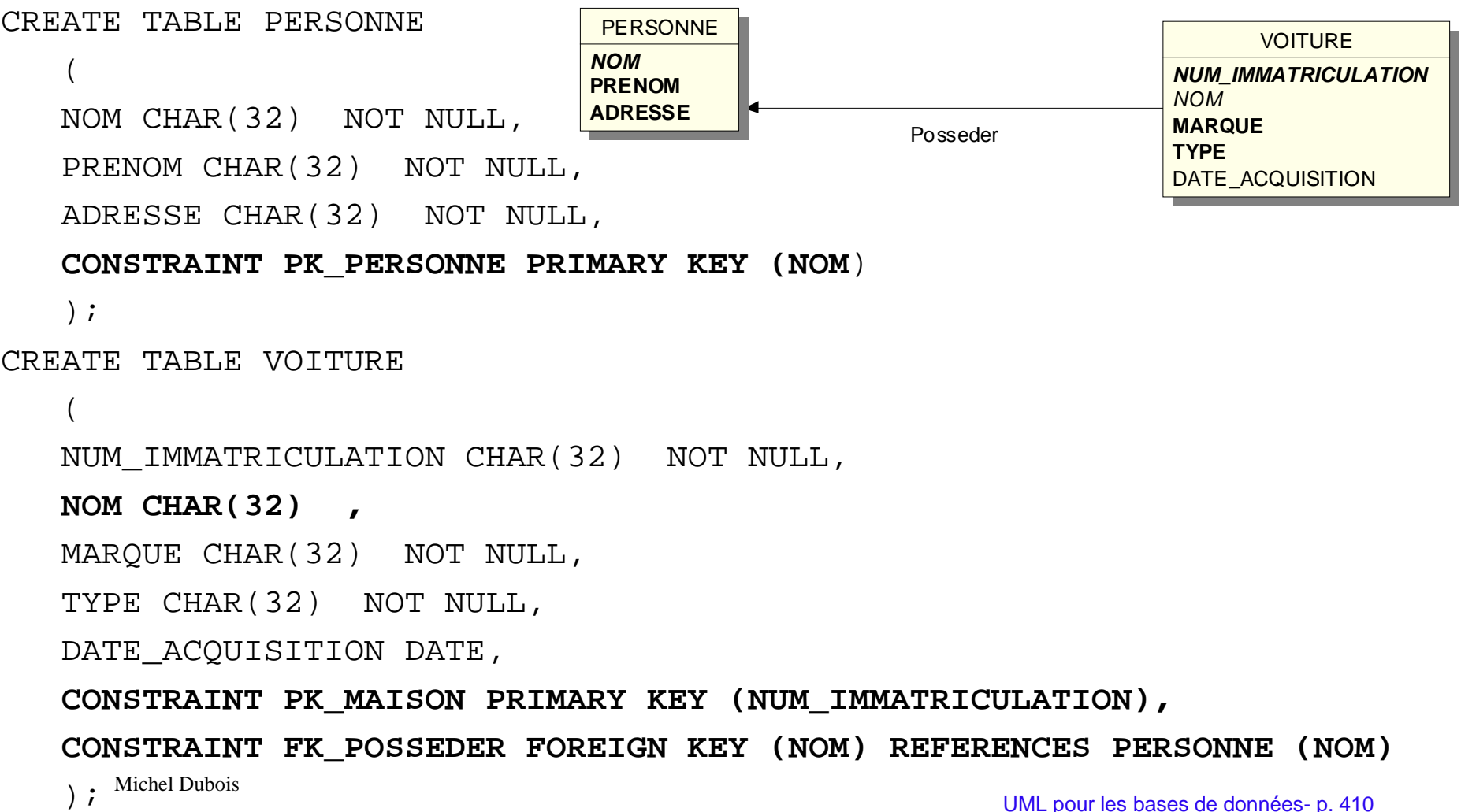


# ASSOCIATION 1:N -> (0,1)-(0,N)

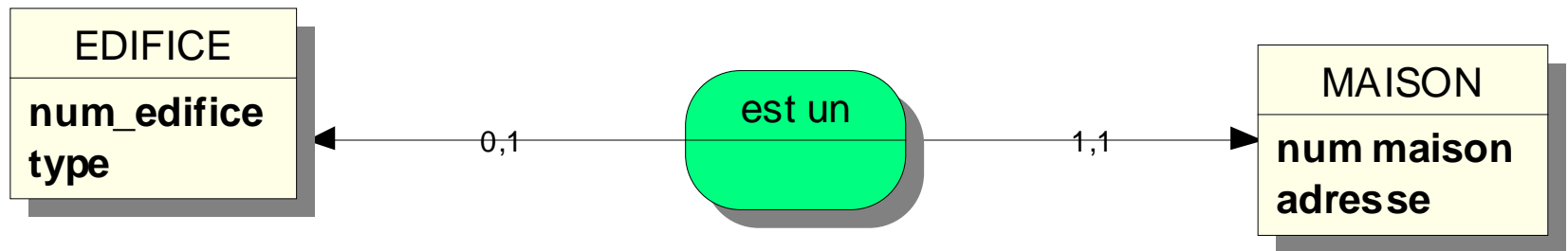
- La règle n°2 s'applique mais il y a d'abord migration de l'attribut date acquisition dans voiture.



# ASSOCIATION 1:N -> (0,1)-(0,N)

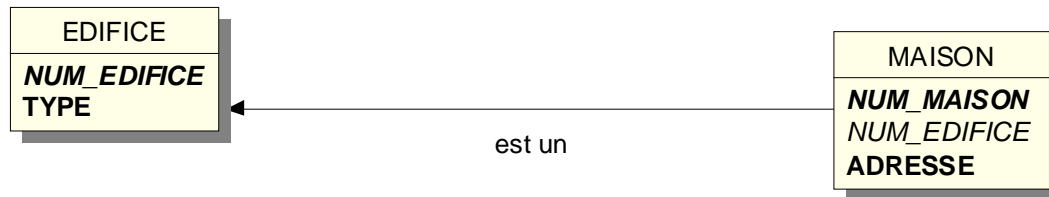


## *ASSOCIATION 1:1 -> (1,1)-(0,1)*



La règle n°2 s 'applique qu 'une seule fois.

On va privilégier la dépendance fonctionnelle forte.



**MAISON.NUM\_EDIFICE** est obligatoire.

**MAISON.NUM\_EDIFICE** n 'accepte pas de doublon.

# *ASSOCIATION 1:1 -> (1,1)-(0,1)*

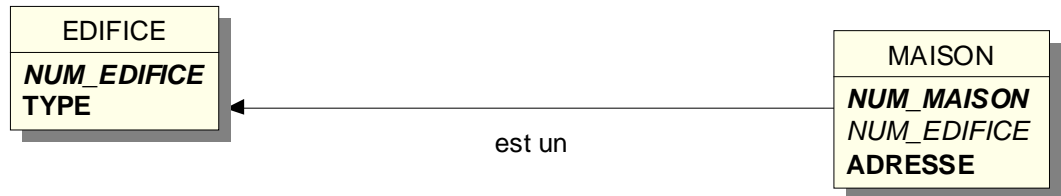
```
CREATE TABLE EDIFICE
```

```
(  
  NUM_EDIFICE CHAR(32) NOT NULL,  
  TYPE CHAR(32) NOT NULL,  
  CONSTRAINT PK_EDIFICE PRIMARY KEY (NUM_EDIFICE)  
);
```

```
CREATE TABLE MAISON
```

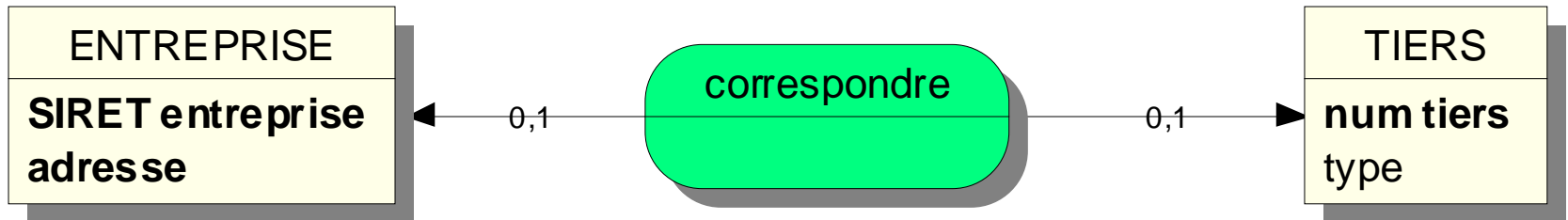
```
(  
  NUM_MAISON NUMBER(4) NOT NULL,  
  NUM_EDIFICE CHAR(32) NOT NULL UNIQUE,  
  ADRESSE CHAR(32) NOT NULL,  
  CONSTRAINT PK_MAISON PRIMARY KEY (NUM_MAISON),  
  CONSTRAINT FK_EST_UN FOREIGN KEY (NUM_EDIFICE) REFERENCES EDIFICE  
  (NUM_EDIFICE)  
);
```

Michel Dubois

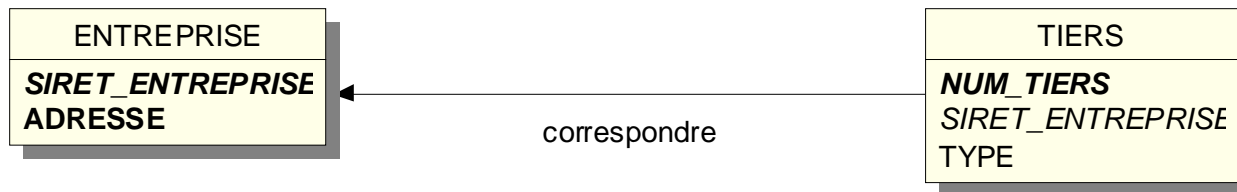




# ASSOCIATION 1:1 -> (0,1)-(0,1)



- La règle n°2 s'applique qu'une seule fois.
- On va privilégier un sens (au choix)



TIERS.SIRET\_ENTREPRISE n'accepte pas de doublon

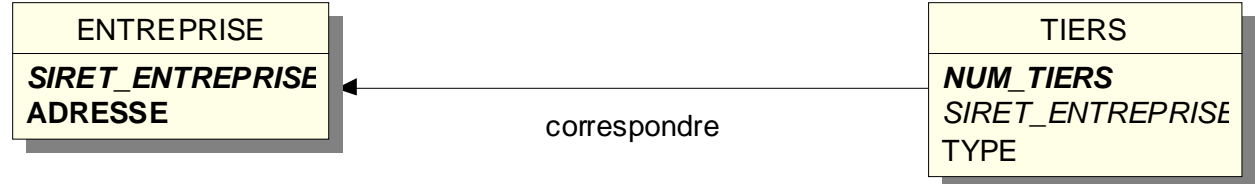
# *ASSOCIATION 1:1 -> (0,1)-(0,1)*

```
CREATE TABLE ENTREPRISE
```

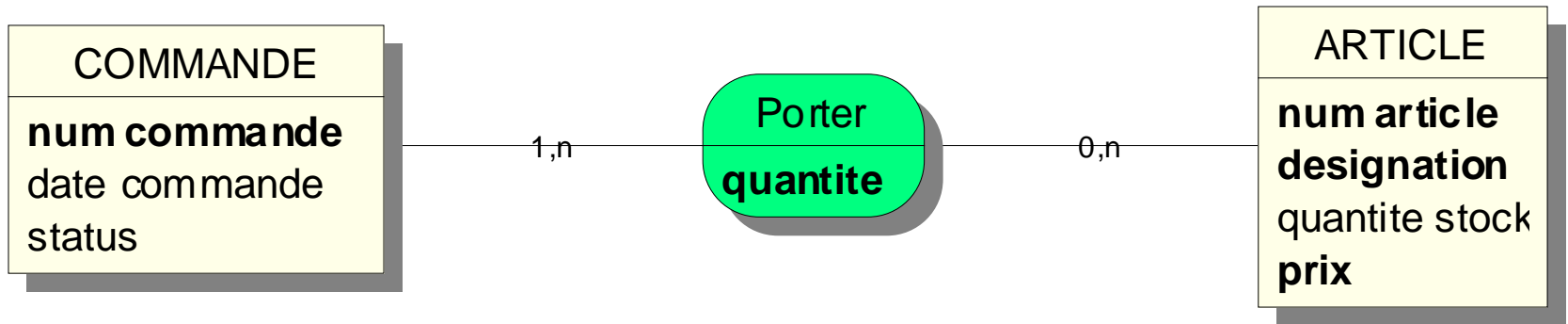
```
(  
  SIRET_ENTREPRISE CHAR(32)  NOT NULL,  
  ADRESSE CHAR(32)  NOT NULL,  
  CONSTRAINT PK_ENTREPRISE PRIMARY KEY (SIRET_ENTREPRISE)  
);
```

```
CREATE TABLE TIERS
```

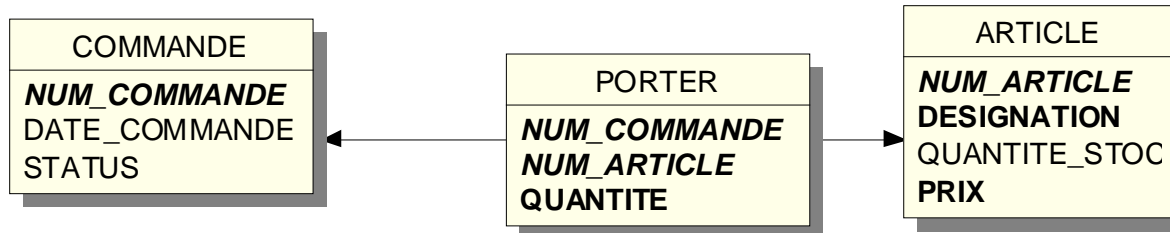
```
(  
  NUM_TIERS CHAR(32)  NOT NULL,  
  SIRET_ENTREPRISE CHAR(32) UNIQUE ,  
  TYPE CHAR(32) ,  
  CONSTRAINT PK_TIERS PRIMARY KEY (NUM_TIERS),  
  CONSTRAINT FK_CORRESPONDRE FOREIGN KEY (SIRET_ENTREPRISE) REFERENCES  
  ENTREPRISE (SIRET_ENTREPRISE)  
);
```



# ASSOCIATION $N:M \rightarrow (X,N)-(X,N)$



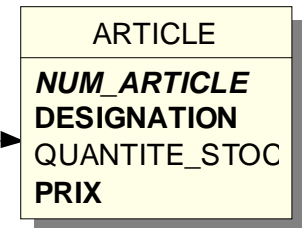
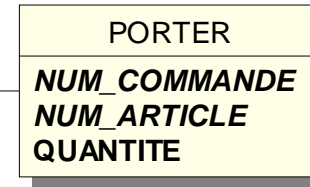
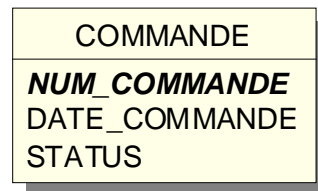
La règle n°3 s'applique.



# ASSOCIATION $N:M \rightarrow (X,N)-(X,N)$

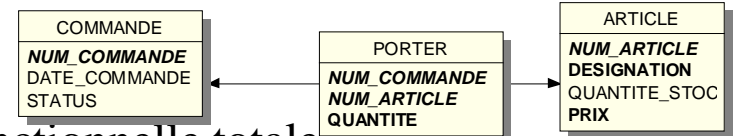
```
CREATE TABLE COMMANDE
(
  NUM_COMMANDE NUMBER(4)  NOT NULL,
  DATE_COMMANDE DATE    ,
  STATUS CHAR(32)      ,
  CONSTRAINT PK_COMMANDE PRIMARY KEY (NUM_COMMANDE)
);
```

```
CREATE TABLE ARTICLE
(
  NUM_ARTICLE CHAR(32)  NOT NULL,
  DESIGNATION CHAR(32) NOT NULL,
  QUANTITE_STOCK NUMBER(4)  ,
  PRIX NUMBER(4)  NOT NULL,
  CONSTRAINT PK_ARTICLE PRIMARY KEY (NUM_ARTICLE)
);
```



```
CREATE TABLE PORTER
(
  NUM_COMMANDE NUMBER(4)  NOT NULL,
  NUM_ARTICLE CHAR(32)  NOT NULL,
  QUANTITE NUMBER(4)  NOT NULL,
  CONSTRAINT PK_PORTER PRIMARY KEY (NUM_COMMANDE, NUM_ARTICLE),
  CONSTRAINT FK_COMMANDE FOREIGN KEY (NUM_COMMANDE) REFERENCES COMMANDE (NUM_COMMANDE),
  CONSTRAINT FK_ARTICLE FOREIGN KEY (NUM_ARTICLE) REFERENCES ARTICLE (NUM_ARTICLE)
);
```

# ASSOCIATION $N:M \rightarrow (1,N)-(0,N)$



- La règle n°3 s'applique.
- Mais il y a aussi une contrainte de dépendance fonctionnelle totale.
  - $\text{Porter.num\_commande} \rightarrow (\text{T}) \text{Commande.num\_commande}$
  - Toute commande de la relation commande se retrouve dans la relation porter.

$\text{COMMANDE}[\text{num\_commande}] = \text{PORTER}[\text{num\_commande}]$

$\Leftrightarrow \text{PORTER}[\text{num\_commande}] \subseteq \text{COMMANDE}[\text{num\_commande}] : \text{FK}$

et  $\text{COMMANDE}[\text{num\_commande}] \subseteq \text{PORTER}[\text{num\_commande}]$

On teste :

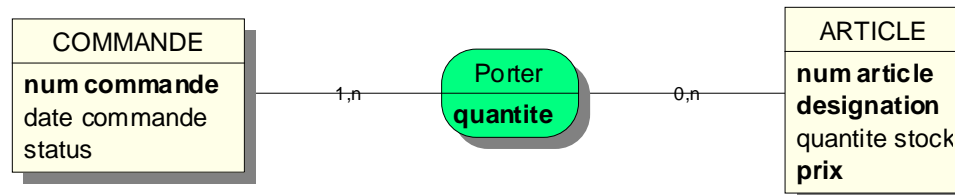
$\text{COMMANDE}[\text{num\_commande}] - \text{PORTER}[\text{num\_commande}] = \emptyset$

...WHERE NOT EXISTS (

SELECT num\_commande FROM commande

MINUS

SELECT num\_commande FROM porter)



# ASSOCIATION $N:M \rightarrow (1,N)-(0,N)$

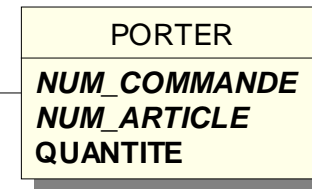
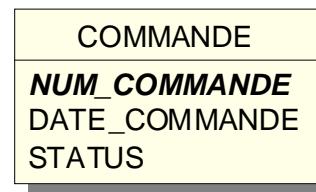
`COMMANDE[num_commande]  $\subseteq$  PORTER[num_commande]`

`CREATE TABLE COMMANDE`

```
(  
  NUM_COMMANDE NUMBER(4) NOT NULL,  
  DATE_COMMANDE DATE ,  
  STATUS CHAR(32) ,  
  CONSTRAINT PK_COMMANDE PRIMARY KEY (NUM_COMMANDE)  
);
```

`CREATE TABLE ARTICLE`

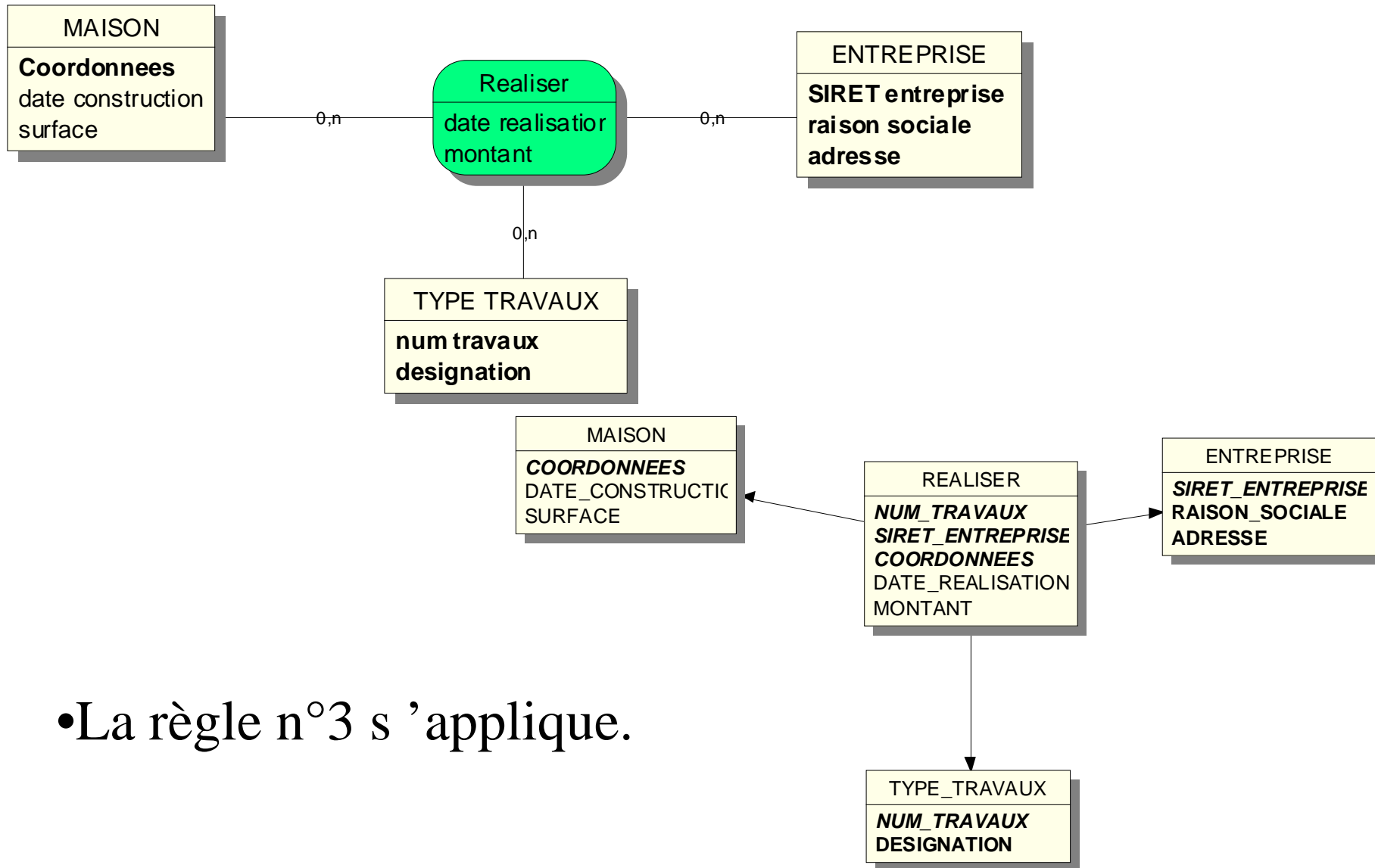
```
(  
  NUM_ARTICLE CHAR(32) NOT NULL,  
  DESIGNATION CHAR(32) NOT NULL,  
  QUANTITE_STOCK NUMBER(4) ,  
  PRIX NUMBER(4) NOT NULL,  
  CONSTRAINT PK_ARTICLE PRIMARY KEY (NUM_ARTICLE)  
);
```



`CREATE TABLE PORTER`

```
(  
  NUM_COMMANDE NUMBER(4) NOT NULL,  
  NUM_ARTICLE CHAR(32) NOT NULL,  
  QUANTITE NUMBER(4) NOT NULL,  
  CONSTRAINT PK_PORTER PRIMARY KEY (NUM_COMMANDE, NUM_ARTICLE),  
  CONSTRAINT FK_COMMANDE FOREIGN KEY (NUM_COMMANDE) REFERENCES COMMANDE (NUM_COMMANDE),  
  CONSTRAINT FK_ARTICLE FOREIGN KEY (NUM_ARTICLE) REFERENCES ARTICLE (NUM_ARTICLE)  
);
```

# ASSOCIATION N-AIRE



•La règle n°3 s 'applique.

# ASSOCIATION N-AIRE

```

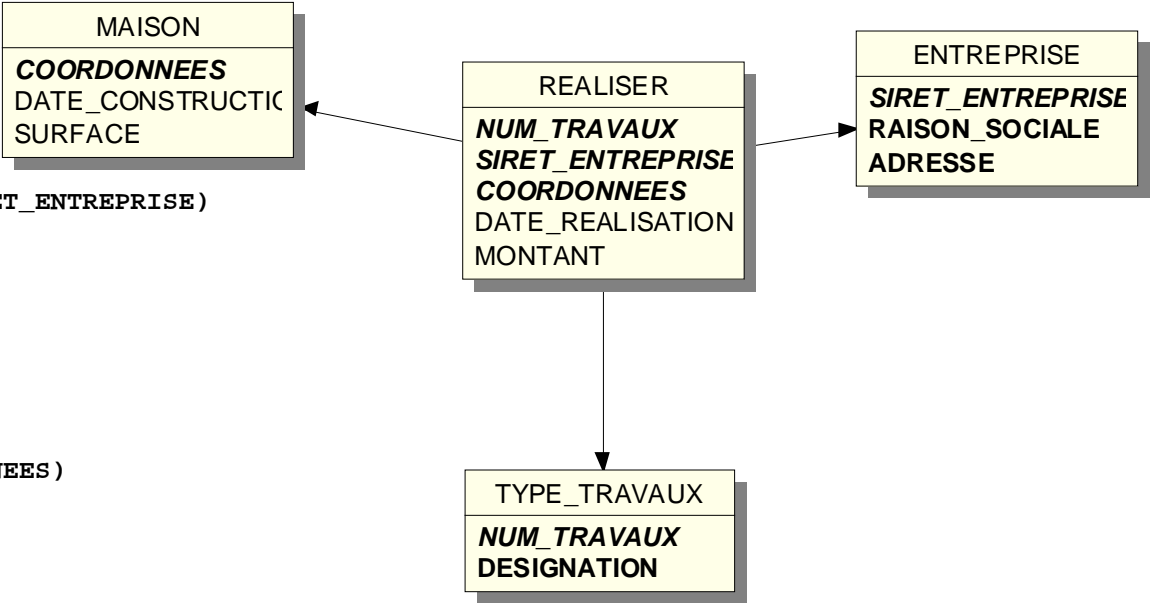
CREATE TABLE ENTREPRISE
(
  SIRET_ENTREPRISE CHAR(32) NOT NULL,
  RAISON_SOCIALE CHAR(32) NOT NULL,
  ADRESSE CHAR(32) NOT NULL,
  CONSTRAINT PK_ENTREPRISE PRIMARY KEY (SIRET_ENTREPRISE)
);

CREATE TABLE MAISON
(
  COORDONNEES CHAR(32) NOT NULL,
  DATE_CONSTRUCTION DATE ,
  SURFACE NUMBER(4),
  CONSTRAINT PK_MAISON PRIMARY KEY (COORDONNEES)
);

CREATE TABLE TYPE_TRAVAUX
(
  NUM_TRAVAUX NUMBER(4) NOT NULL,
  DESIGNATION CHAR(32) NOT NULL,
  CONSTRAINT PK_TYPE_TRAVAUX PRIMARY KEY (NUM_TRAVAUX)
);

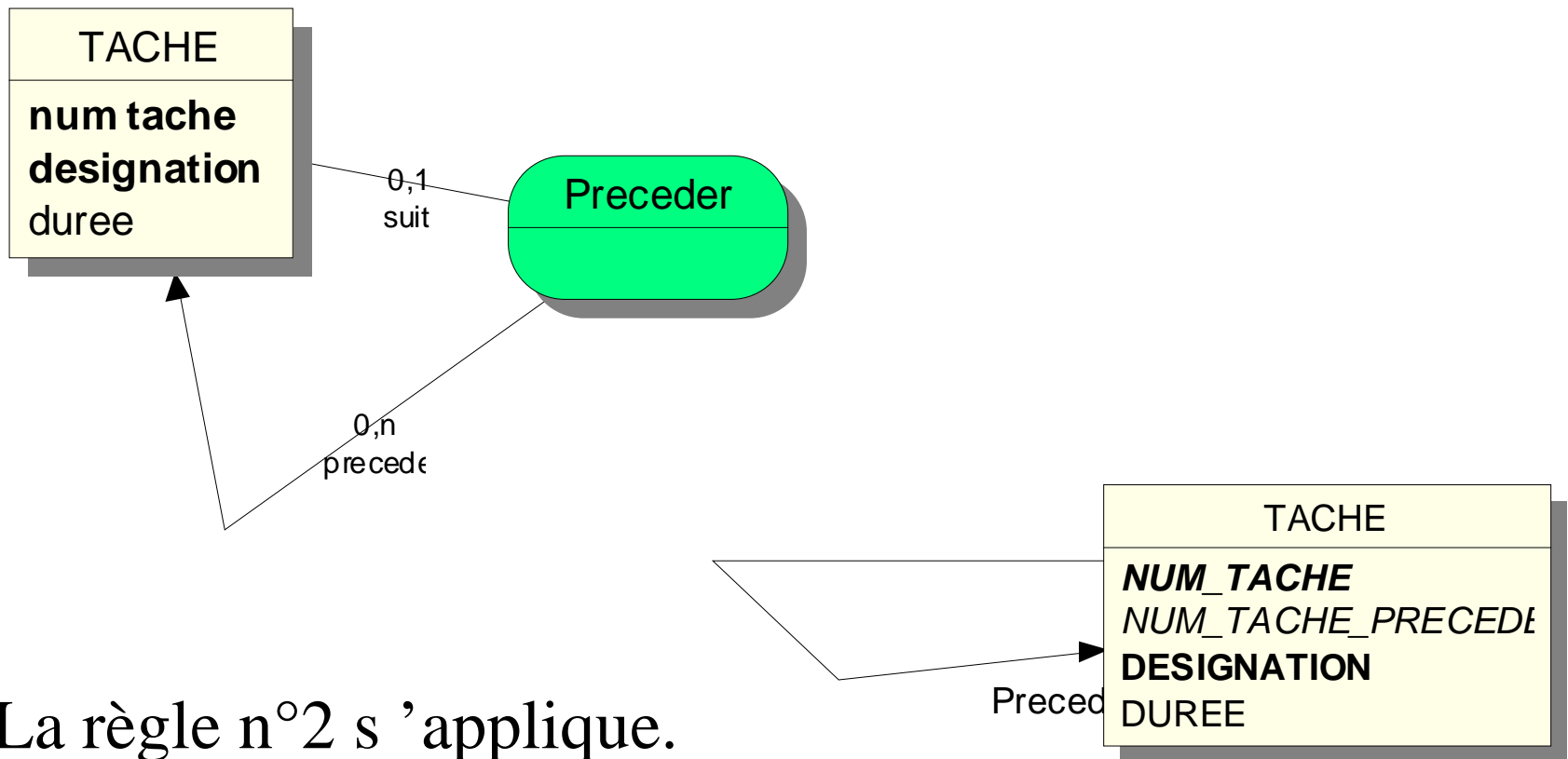
CREATE TABLE REALISER
(
  NUM_TRAVAUX NUMBER(4) NOT NULL,
  SIRET_ENTREPRISE CHAR(32) NOT NULL,
  COORDONNEES CHAR(32) NOT NULL,
  DATE_REALISATION DATE ,
  MONTANT NUMBER(4),
  CONSTRAINT PK_REALISER PRIMARY KEY (NUM_TRAVAUX, SIRET_ENTREPRISE, COORDONNEES),
  CONSTRAINT FK_TYPE_TRAVAUX FOREIGN KEY (NUM_TRAVAUX) REFERENCES TYPE_TRAVAUX (NUM_TRAVAUX),
  CONSTRAINT FK_ENTREPRISE FOREIGN KEY (SIRET_ENTREPRISE) REFERENCES ENTREPRISE (SIRET_ENTREPRISE),
  CONSTRAINT FK_MAISON FOREIGN KEY (COORDONNEES) REFERENCES MAISON (COORDONNEES)
);

```





# ASSOCIATION 1:N REFLEXIVE

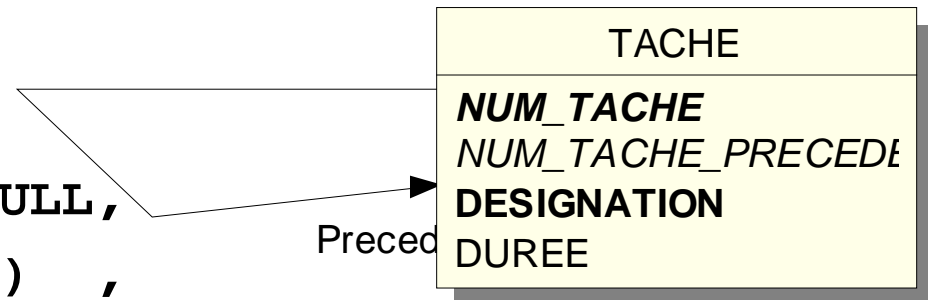


•La règle n°2 s 'applique.

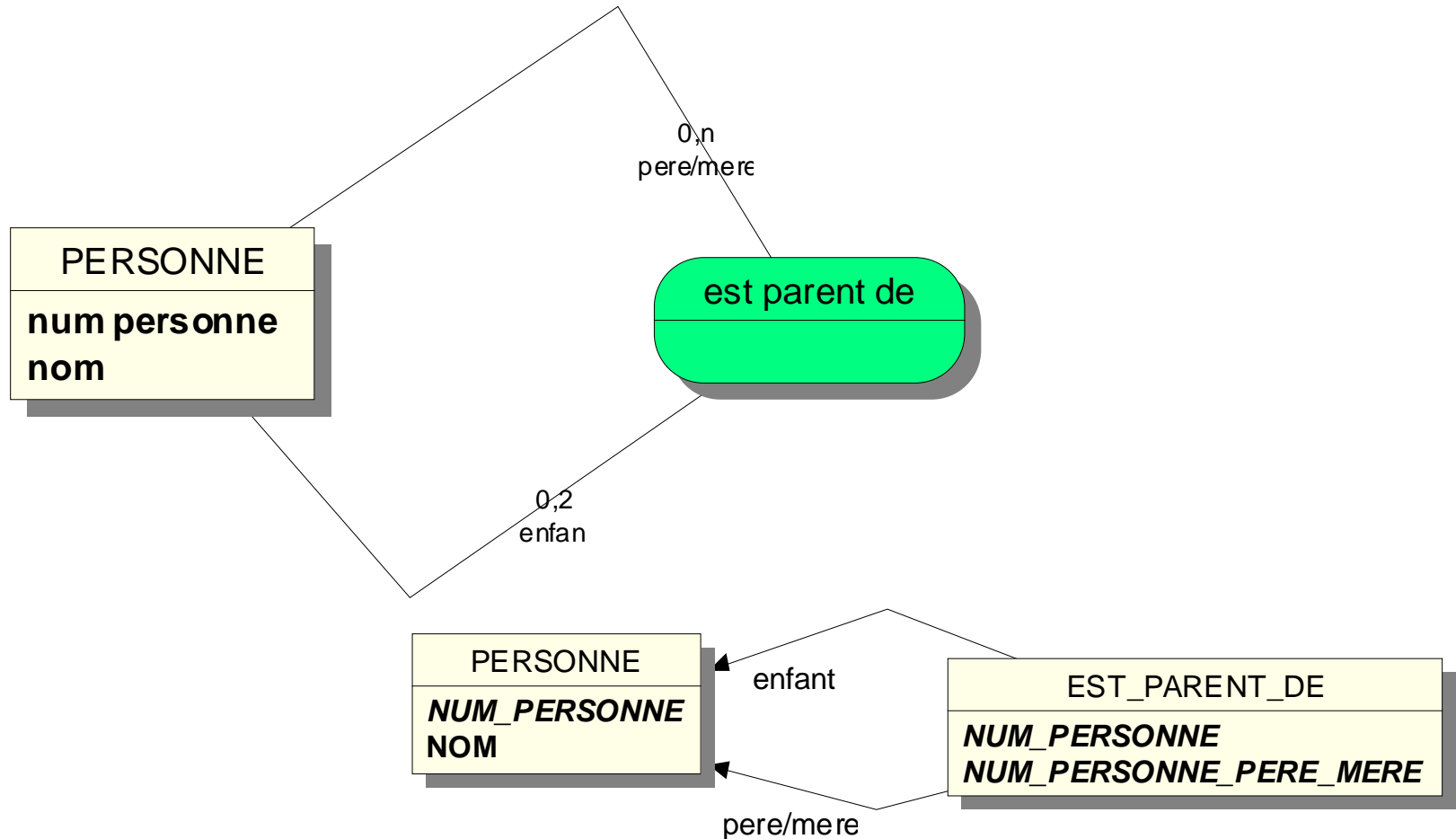
# ASSOCIATION 1:N REFLEXIVE

```
CREATE TABLE TACHE
```

```
(  
  NUM_TACHE NUMBER(4) NOT NULL,  
  NUM_TACHE_PRECEDE NUMBER(4) ,  
  DESIGNATION CHAR(32) NOT NULL,  
  DUREE DATE ,  
  CONSTRAINT PK_TACHE PRIMARY KEY (NUM_TACHE) ,  
  CONSTRAINT FK_PRECEDE FOREIGN KEY (NUM_TACHE_PRECEDE)  
  REFERENCES TACHE (NUM_TACHE)  
);
```



# ASSOCIATION N:M REFLEXIVE



- La règle n°3 s'applique.

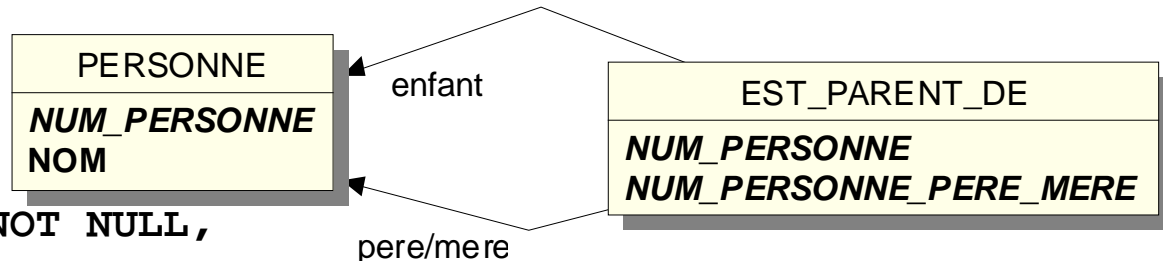
# ASSOCIATION N:M REFLEXIVE

```
CREATE TABLE PERSONNE
```

```
(  
  NUM_PERSONNE NUMBER(4) NOT NULL,  
  NOM CHAR(32) NOT NULL,  
  CONSTRAINT PK_PERSONNE PRIMARY KEY (NUM_PERSONNE)  
);
```

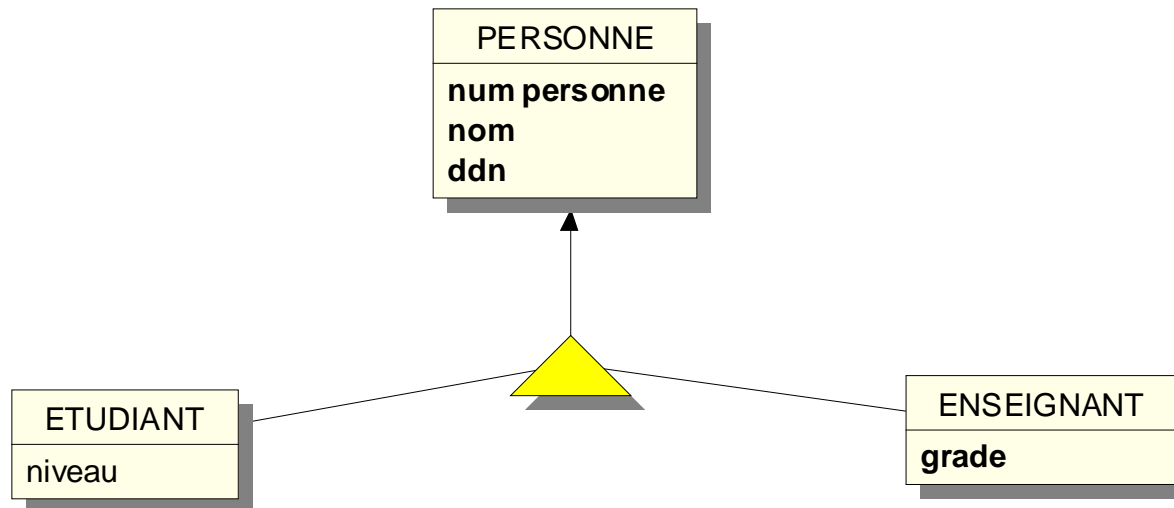
```
CREATE TABLE EST_PARENT_DE
```

```
(  
  NUM_PERSONNE NUMBER(4) NOT NULL,  
  NUM_PERSONNE_PERE_MERE NUMBER(4) NOT NULL,  
  CONSTRAINT PK_EST_PARENT_DE PRIMARY KEY (NUM_PERSONNE,  
  NUM_PERSONNE_PERE_MERE),  
  CONSTRAINT FK_PERSONNE FOREIGN KEY (NUM_PERSONNE) REFERENCES PERSONNE  
  (NUM_PERSONNE),  
  CONSTRAINT FK_PERE_MERE FOREIGN KEY (NUM_PERSONNE_PERE_MERE)  
  REFERENCES PERSONNE (NUM_PERSONNE)  
);
```



# *SPÉCIALISATION*

- La règle n°2 peut s'appliquer.

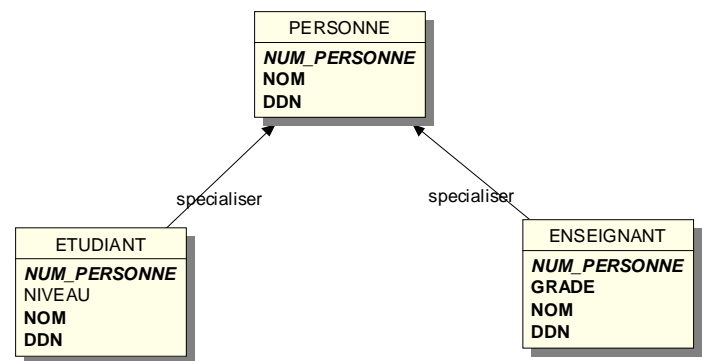


# SPÉCIALISATION

```
CREATE TABLE PERSONNE
(
  NUM_PERSONNE NUMBER(4) NOT NULL,
  NOM CHAR(32) NOT NULL,
  DDN DATE NOT NULL,
  CONSTRAINT PK_PERSONNE PRIMARY KEY (NUM_PERSONNE)
);

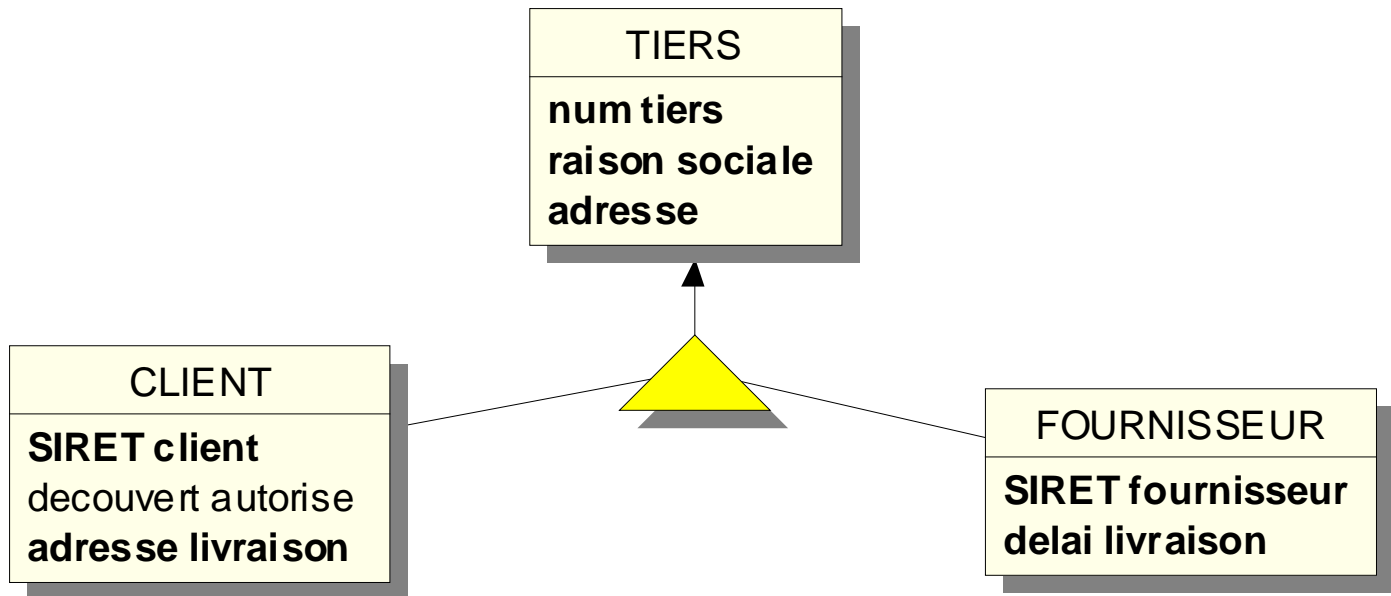
CREATE TABLE ETUDIANT
(
  NUM_PERSONNE NUMBER(4) NOT NULL,
  NIVEAU CHAR(32) ,
  NOM CHAR(32) NOT NULL,
  DDN DATE NOT NULL,
  CONSTRAINT PK_ETUDIANT PRIMARY KEY (NUM_PERSONNE),
  CONSTRAINT FK_SPECIALISER1 FOREIGN KEY (NUM_PERSONNE) REFERENCES PERSONNE (NUM_PERSONNE)
);

CREATE TABLE ENSEIGNANT
(
  NUM_PERSONNE NUMBER(4) NOT NULL,
  GRADE CHAR(32) NOT NULL,
  NOM CHAR(32) NOT NULL,
  DDN DATE NOT NULL,
  CONSTRAINT PK_ENSEIGNANT PRIMARY KEY (NUM_PERSONNE),
  CONSTRAINT FK_SPECIALISER2 FOREIGN KEY (NUM_PERSONNE) REFERENCES PERSONNE (NUM_PERSONNE)
);
```



# GÉNÉRALISATION

- La règle n°2 peut s'appliquer.

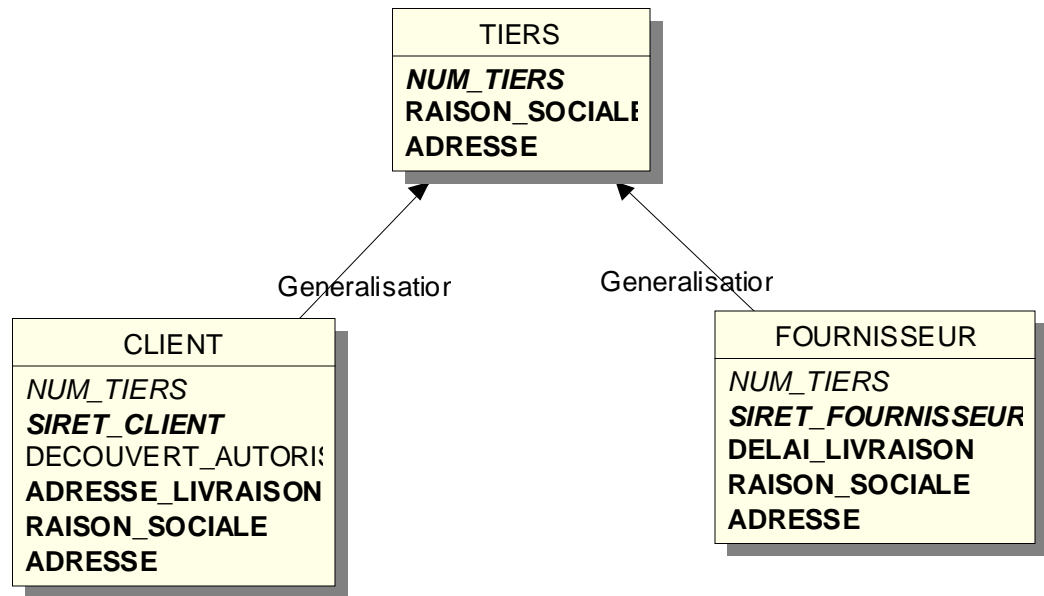


# GÉNÉRALISATION

```
CREATE TABLE TIERS
(
  NUM_TIERS CHAR(32) NOT NULL,
  RAISON_SOCIALE CHAR(32) NOT NULL,
  ADRESSE CHAR(32) NOT NULL,
  CONSTRAINT PK_TIERS PRIMARY KEY (NUM_TIERS)
);

CREATE TABLE CLIENT
(
  NUM_TIERS CHAR(32) NOT NULL,
  SIRET_CLIENT CHAR(32) NOT NULL,
  DECOUVERT_AUTORISE NUMBER(4) ,
  ADRESSE_LIVRAISON CHAR(32) NOT NULL,
  RAISON_SOCIALE CHAR(32) NOT NULL,
  ADRESSE CHAR(32) NOT NULL,
  CONSTRAINT PK_CLIENT PRIMARY KEY (SIRET_CLIENT),
  CONSTRAINT FK_GENERALISATION1 FOREIGN KEY (NUM_TIERS) REFERENCES TIERS (NUM_TIERS)
);

CREATE TABLE FOURNISSEUR
(
  NUM_TIERS CHAR(32) NOT NULL,
  SIRET_FOURNISSEUR CHAR(32) NOT NULL,
  DELAI_LIVRAISON DATE NOT NULL,
  RAISON_SOCIALE CHAR(32) NOT NULL,
  ADRESSE CHAR(32) NOT NULL,
  CONSTRAINT PK_FOURNISSEUR PRIMARY KEY (SIRET_FOURNISSEUR),
  CONSTRAINT FK_GENERALISATION2 FOREIGN KEY (NUM_TIERS) REFERENCES TIERS (NUM_TIERS)
);
```







# Chapitre 10

## ORM

### Correspondance

### Objet - Relationnel

Michel Dubois

I.U.T. de Vannes

*Michel.Dubois@univ-ubs.fr*

[Retour au plan](#)



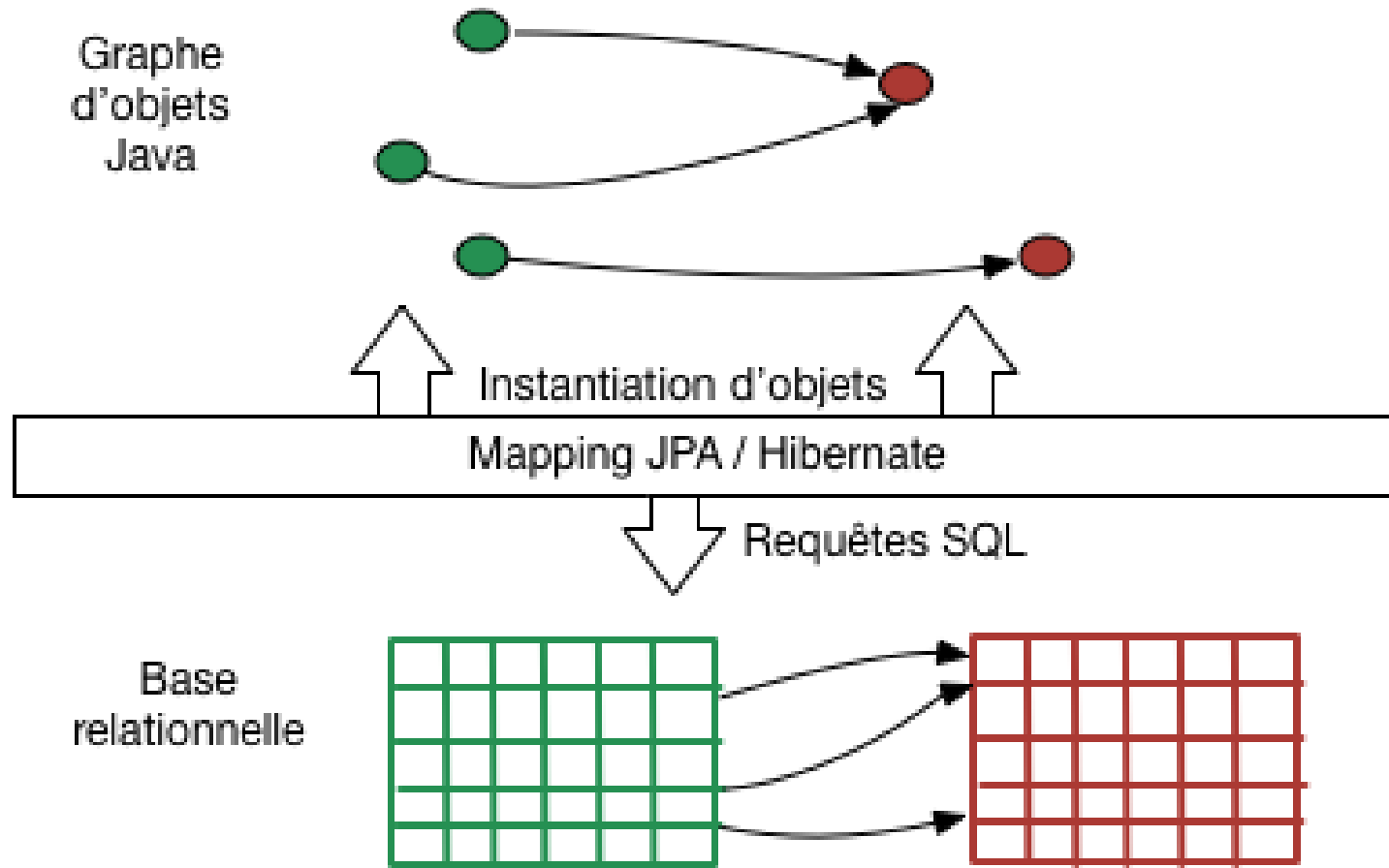
# Bibliographie

- Hibernate
  - «*JPA et Java Hibernate - Apprenez le mapping objet-relationnel (ORM) avec Java*», Martial BANON, ENI, février 2017. Accessible via le [web](#),/

# Incompatibilité relationnel et objet

- Les représentations utilisées dans la base de données et dans une application objet sont incompatibles :
  - Le rôle d'un système ORM est de convertir automatiquement, à la demande, la base de données sous forme d'un graphe d'objet.
  - L'ORM s'appuie pour cela sur une configuration associant les classes du modèle fonctionnel et le schéma de la base de données.
  - L'ORM génère des requêtes SQL qui permettent de matérialiser ce graphe ou une partie de ce graphe en fonction des besoins.

# Mapping d'une BD relationnelle en graphe d'objets Java





# Java Persistence API ou JPA

- JPA est essentiellement une spécification intégrée au JEE qui vise à standardiser la couche d'association entre une base relationnelle et une application Java construite sur des objets.
- JPA n'est qu'une spécification visant à la standardisation
- JPA définit une interface dans le package `javax.persistence.*`.
- La définition des associations avec la base de données s'appuie essentiellement sur les annotations Java, évite d'avoir de longs fichiers de configuration XML qui doivent être maintenus en parallèle aux fichiers de code.
- On obtient une manière assez légère de définir une sorte de base de données objet virtuelle qui est matérialisée au cours de l'exécution d'une application par des requêtes SQL produites par le framework sous-jacent.

## Deux formes de mapping

### Hibernate : Mapping par fichier

#### XML

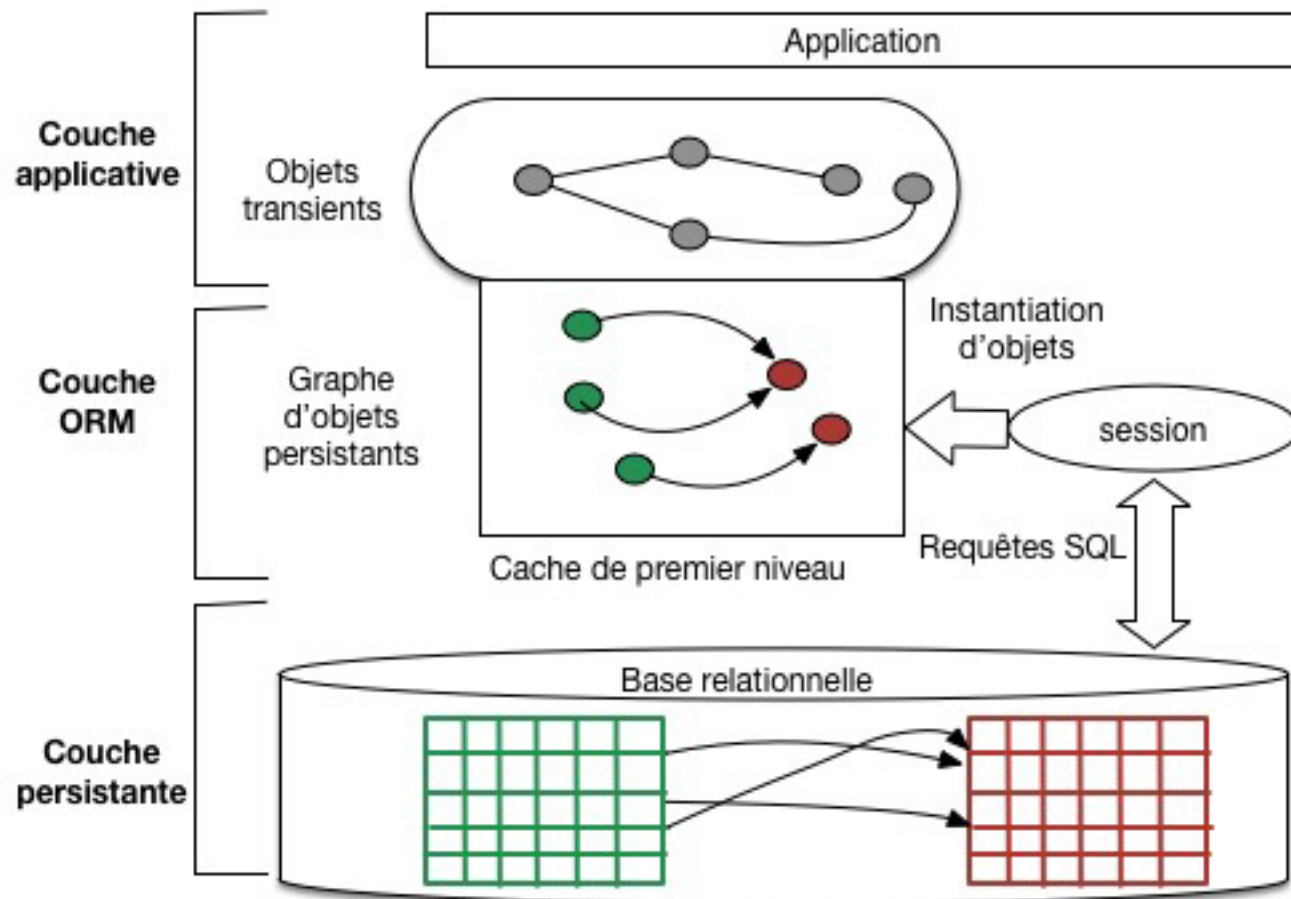
```
<class name="person" table="PERSON">
<id name="id" column="PERSON_ID">
  <generator class="foreign">
    <param name="property">employee</param>
  </generator>
</id>
  <one-to-one name="employee"
    class="Employee" constrained="true"/>
</class>
```

[francois.jannin@inp-toulouse.fr](mailto:francois.jannin@inp-toulouse.fr)

### JPA/EJB3 : Mapping par annotations Java

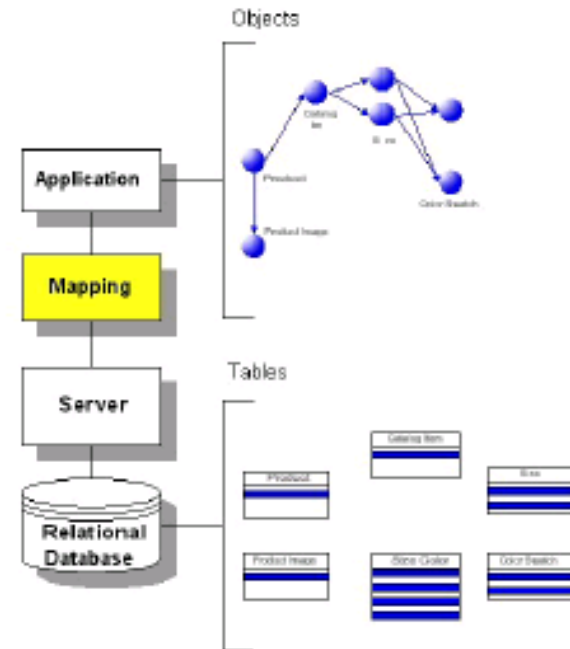
```
@Entity
@Table(name="Personne")
public class Personne implements Serializable{
  @Id
  @Column(name = "ID", nullable = false)
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Integer id;
  @Column(name = "NOM", length = 30, nullable = false,
    unique = true)
  private String nom;
  ...
  // constructeurs
  public Personne() { }
  ...
```

# Structure en couches une application avec ORM



# Non correspondance des modèles objet et relationnel

- *Impedance mismatch*
- Entrée : modèle objet
- Sortie : modèle relationnel
- Problèmes de correspondance :
  - Identification des objets
  - Traduction des associations
  - Traduction de l'héritage
  - Navigation entre les objets
  - Objets dépendants







# Opérations de lecture

- les différentes options de lecture de données :
  - par navigation dans le graphe des objets, si nécessaire chargé à la volée par Hibernate; par identifiant: méthode basique, rapide, et de fait utilisée implicitement par d'autres méthodes comme la navigation;
  - par le langage de requêtes HQL;
  - par l'API Criteria, qui permet de construire par programmation objet une requête à exécuter;
  - enfin, directement par SQL, ce qui n'est pas une méthode portable et devrait donc être évité.
  - JPA définit un langage de requête, JPQL (Java Persistence Query Language) qui est un sous-ensemble de HQL. Toute requête JPQL est une requête HQL, l'inverse n'est pas vrai.



# ORM et Eclipse

- Visual Paradigm
  - Tools / DB et Tools / ORM
- Web Tools Plateform (projet DALI pour JPA)
  - New Project,,, -> JPA -> JPA Project
- Jboss Tools (Hibernate Tools)
  - A installer avec le site update pour Eclipse Juno
  - Générateur de configuration
  - Générateur de code
  - Editeur de requête HQL
- Liferay IDE avec le Service Builder

## Hibernate Tools Configuration : guide rapide

Lancez tout d'abord la fenêtre *Hibernate Configurations*.

Vous pouvez créer des configurations (fichier *hibernate.cfg.xml*).

Une configuration peut être créée à partir d'un projet existant, ce qui revient à utiliser les paramètres Hibernate déjà définis pour le projet.

Quand vous êtes sur l'onglet *Configuration* de la nouvelle fenêtre, des boutons sur la droite permettent de créer une nouvelle configuration

Pour une configuration existante, le bouton droit donne accès à l'édition de la configuration, au lancement d'un éditeur HQL, etc.

Le plus simple est donc d'indiquer le projet associé à la configuration, et d'utiliser le fichier de *mapping hibernate.cfg.xml* de ce projet.

Cela assure que les classes persistantes et la connexion à la base sont automatiquement détectées.

L'outil vous permet également de créer ou modifier un fichier de configuration avec une interface graphique assez agréable.

# Configuration d'une session d'Hibernate

The screenshot shows the 'Edit Configuration' dialog box for a console configuration named 'hibernate'. The dialog has a title bar with standard window controls and a green play button icon. Below the title bar, the text 'Edit launch configuration properties' and 'Select or configure a Console Configuration' is displayed. The main area is divided into tabs: 'Main' (selected), 'Options', 'Classpath', 'Mappings', and 'Common'. Under the 'Main' tab, the 'Type' section has three radio buttons: 'Core', 'Annotations (jdk 1.5+)' (selected), and 'JPA (jdk 1.5+)'. Below this is a 'Hibernate Version' dropdown set to '4.0'. The 'Project' section has a text field containing 'orm' and a 'Browse...' button. The 'Database connection' section has a dropdown menu showing '[Hibernate configured connection]', a 'New...' button, and an 'Edit...' button. The 'Property file' section has an empty text field and a 'Setup...' button. The 'Configuration file' section has a text field containing '/orm/src/hibernate.cfg.xml' and a 'Setup...' button. The 'Persistence unit' section is empty. At the bottom right, there are 'Apply' and 'Revert' buttons. At the bottom left, there is a question mark icon. At the bottom right, there are 'Cancel' and 'OK' buttons.

**Edit Configuration**

**Edit launch configuration properties**  
Select or configure a Console Configuration

Name: **hibernate**

**Main** | Options | Classpath | Mappings | Common

Type:

☐ Core ☒ Annotations (jdk 1.5+) ☐ JPA (jdk 1.5+)

Hibernate Version: **4.0**

Project:

**orm** **Browse...**

Database connection:

**[Hibernate configured connection]** **New...** **Edit...**

Property file:

**Setup...**

Configuration file:

**/orm/src/hibernate.cfg.xml** **Setup...**

Persistence unit:

**Apply** **Revert**

**Cancel** **OK**

## Utilisation

Une fois la configuration créée, vous pouvez lancer un éditeur HQL et y saisir des requêtes.

Pour l'éditeur HQL, sélectionnez votre configuration de session, et utilisez soit le bouton droit, soit l'icône "HQL" que vous voyez sur la barre d'outils.

Vous pouvez saisir des requêtes HQL dans l'éditeur, et les exécuter avec la flèche verte.

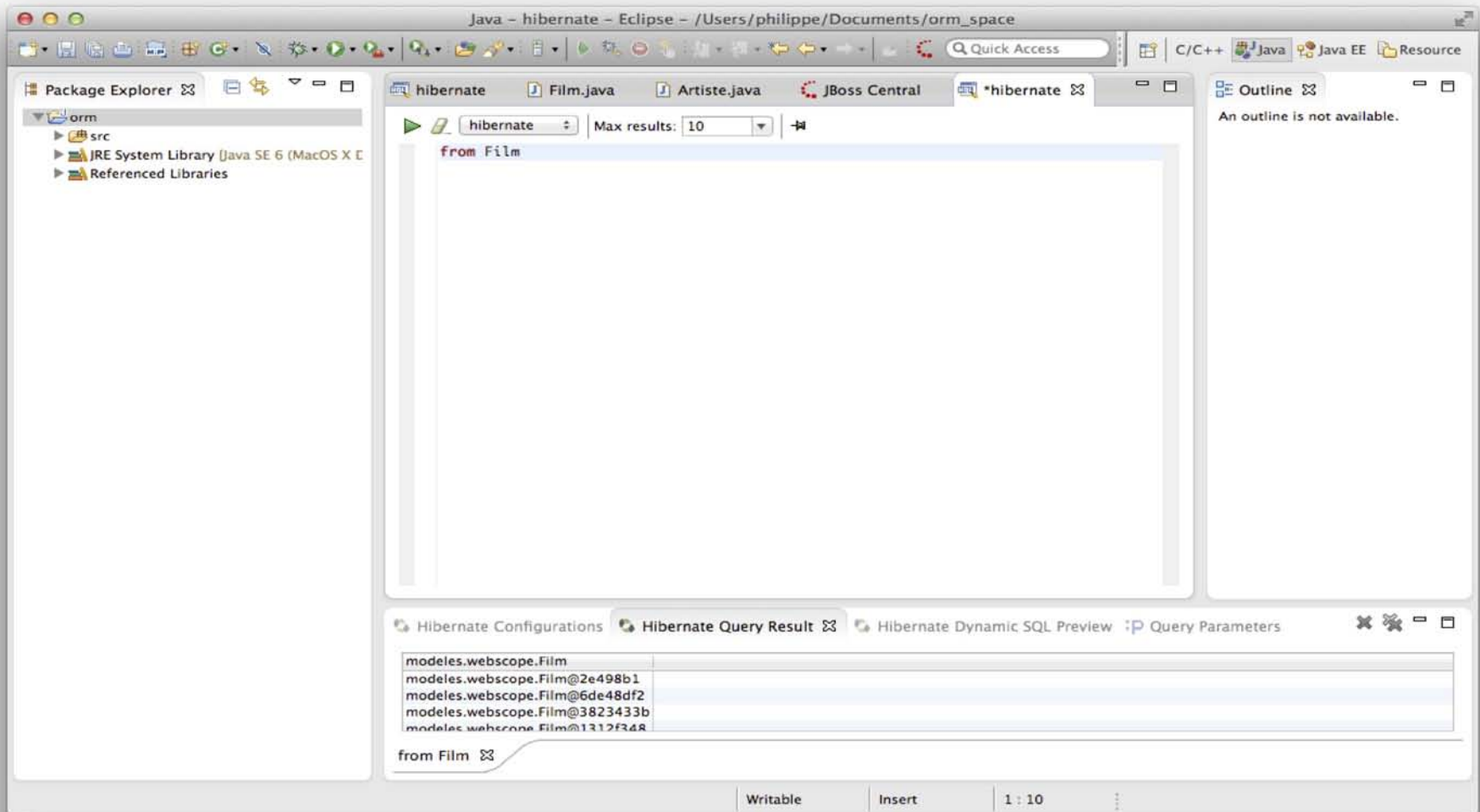
Trois autres vues sont utiles:

*Hibernate Dynamic SQL* Preview vous montre la/les requête(s) SQL lancée(s) par Hibernate pour évaluer la requête HQL (très utile pour comprendre comment Hibernate matérialise le graphe requis par une requête HQL)

*Hibernate Query Result* vous donne simplement le résultat de la requête HQL.

*Query parameters* sert à définir la valeur des paramètres pour les requêtes qui en comportent.

# Les fenêtres Hibertools



## Utilisation

les requêtes HQL renvoient en général des *objets*

Hibernate Tools affiche la *référence* de ces objets (peu parlant)

Pour consulter plus facilement le résultat des requêtes, on peut ajouter une clause *select* (optionnelle en HQL) :

---

```
select film.titre, film.annee  
from Film as film
```

---

## HQL et Java

L'intégration de HQL à Java se fait par des méthodes de la *Session*. Le code suivant recherche un (ou plusieurs) film(s) par leur titre.

---

```
public List<Film> parTitre(String titre)
{
    Query q = session.createQuery("from Film f where f.titre= :titre"); q.setString
    ("titre", titre);
    return q.list();
}
```

---

La clause *select* est optionnelle en HQL

elle offre peu d'intérêt dans une application java car on cherche en général à récupérer automatiquement des instances des classes *mappées*: cela implique de récupérer tous les attributs de chaque ligne.



## HQL et Java (2)

Comme en JDBC, on peut introduire dans la requête des *paramètres* en les préfixant par ":" ("?" est également accepté).

Hibernate se charge de protéger la syntaxe de la requête (en ajoutant des barres obliques devant les apostrophes et autres caractères réservés)

Voici une requête un peu plus complète (et plus concise) :

---

```
session.createQuery("from Film as film where film.titre like :titre and film.annee < :an  
.setString ("titre", "%er%")  
.setInteger("annee", 2000)  
.list();
```

---

On applique la technique dite de *chaînage des méthodes* :  
chaque méthode *set* renvoie l'objet-cible  
objet auquel on peut donc appliquer une nouvelle méthode  
et ainsi de suite

Notez également que l'affectation des paramètres tient compte de leur type:  
Hibernate propose des *setDate()*, *setInteger()*, *setTimestamp()*, etc.

On peut également utiliser comme paramètre un objet persistant, comme le montre l'exemple suivant:

---

```
// bergman est une instance de Artiste
Artiste bergman = ...;

session.createQuery("from Film as film where film.realisateur= :mes")
.setEntity ("mes", bergman)
.list();
```

---

HQL offre la possibilité de *paginer* les résultats, une option souvent utile dans un contexte d'application web.

L'exemple suivant retourne les lignes 10 à 19 du résultat.

---

```
session.createQuery("from Film")
.setFirstResult (10)
.setMaxResults(10)
.list();
```

---

## HQL et Java (4)

La méthode *list()* de l'objet *query* est la plus générale pour exécuter une requête et constituer le résultat.

Si vous êtes sûr que ce dernier ne contient qu'un seul objet, vous pouvez utiliser *uniqueResult()* :

---

```
session.createQuery("from Film where titre='Vertigo'")  
.uniqueResult();
```

---

Attention, une exception est levée si plus d'une ligne est trouvée.

Cette méthode ne devrait être appliquée que pour des recherches portant sur des attributs déclarés *uniques* dans le schéma de la base de données (dont la clé primaire).

## Introduction à la sélection d'objets en HQL

HQL est un langage de requêtes **objet** qui sert à interroger le graphe (virtuel au départ) des objets java constituant la vue orientée-objet de la base.

On interroge **un ensemble d'objets java** liés par des associations, et pas directement la base relationnelle qui permet de les matérialiser.

Hibernate se charge d'effectuer les requêtes SQL pour matérialiser la partie du graphe qui satisfait la requête.

Cela donne une syntaxe et une interprétation parfois différente de SQL, même si les deux langages sont en apparence très proches.

Attention : Contrairement à un "vrai" langage d'interrogation orienté-objet, HQL ne permet pas d'appeler les méthodes des classes du modèle.

## La clause **from**

La forme la plus basique de requête HQL comprend la clause "from", et indique la classe dont l'extension est sujet de la recherche :

**from** Film

en cas de hiérarchie d'héritage, il est possible d'interroger les super- et sous-classes.

**from** Video

et la liste des films (sous-classe de "Video") par :

**from** FilmV

## La clause **from** (2)

On peut rechercher tous les objets persistants avec :

```
from java.lang.Object
```

On peut créer des **alias** pour désigner les objets

on peut ensuite leur appliquer des traitements de sélection ou projection

---

```
select film.titre  
from Film as film  
where film.annee < 2000
```

---

Le mot-clé "as" est optionnel (comme la clause "select"), mais on le garde dans un souci de clarté

Préfixer un attribut par l'alias n'est indispensable que s'il y a risque d'ambiguïté, ce qui n'est pas le cas dans l'exemple ci-dessus.

## Clause **where**

Les opérateurs de comparaison sont très proches de ceux de SQL :

= < > **between in not between**

Les opérateurs “like” et “not like” s'utilisent comme en SQL et expriment la même chose.

Les opérateurs logiques sont les mêmes qu'en SQL: “and”, “or” et “not”, avec ordre de priorité défini par le parenthésage.

il y a aussi une clause **order by** :

---

```
from Film as film
where (titre like '%er%' and annee between 1970 and 2000)
or film.genre='Drame'
order by film.titre
```

---

## Clause **where**

Attention à la valeur "NULL" ou "null".

Comme en SQL, elle indique l'**absence** de valeur.

On la teste avec l'expression "is null", comme en SQL:

---

```
from Film as film  
where film.resume is null
```

---

il existe beaucoup d'autres fonctions pré-définies qui peuvent être appliquées aux propriétés des objets (ex : size(), minElement(), trunc()) (cf doc)



## Jointures implicites

Contrairement à SQL où les critères de sélection ne peuvent concerner que les propriétés de la table interrogée, HQL permet la navigation vers des objets associés (et donc vers d'autres tables), avec une syntaxe abrégée et intuitive.

On peut par exemple chercher les films dirigés par Clint Eastwood:

---

```
from Film as film  
where film.realisateur.nom = 'Eastwood'
```

---

Pas de magie ici : cette syntaxe nécessite une **jointure** au niveau de la requête SQL générée par Hibernate

cela correspond à une manière implicite d'exprimer une jointure

voici la requête générée :

---

```
select *  
from Film film0_ cross join Artiste artiste1_  
where film0_.id_realisateur=artiste1_.id  
and artiste1_.nom='Eastwood'
```

---

## Jointures implicites (2)

Cela ne fonctionne **que** pour les associations *many-to-one* et *one-to-one*.

Vous ne pouvez pas écrire :

---

```
from Film as film  
where film.roles.acteur.nom = 'Eastwood'
```

---

On peut bien entendu exprimer **explicitement** des jointures, avec des options assez riches comme on le verra dans la section suivante.

## Jointures HQL et jointures SQL

Jusqu'à présent, nous prenons les objets d'une classe, ce qui correspond donc à une recherche de ligne dans une seule table pour les instancier.

Comme en SQL, il est nécessaire d'exprimer des **jointures** si l'on veut exprimer des critères de recherche sur les objets associés.

Comme en SQL, une jointure s'exprime avec le "join" :

- en HQL, on exprime des jointures par **navigation** en exploitant les associations d'objets définies dans le **mapping**,

- en SQL, on applique (conceptuellement) le produit cartésien de deux tables, et on restreint le résultat à des combinaisons de lignes satisfaisant certains critères.

**une jointure en HQL qui, en apparence, accède directement aux objets associés, est en fait implantée par Hibernate comme une jointure SQL**

## Jointures HQL et jointures SQL (suite)

Exemple : on veut tous les films mis en scène par Clint Eastwood :

---

```
select film
from Film as film
join film.realisateur as a
where a.nom='Eastwood'
```

---

On accède donc directement à l'objet associé "realisateur", sans mention explicite de la classe "Artiste".

En sous-main, Hibernate engendre et exécute la requête SQL suivante.

---

```
select film.*
from Film film0_ inner join Artiste artiste1_
on film0_.id_realisateur=artiste1_.id
where artiste1_.nom='Eastwood'
```

---

La version SQL comprend une contrainte d'égalité entre la clé primaire de "Artiste" et la clé étrangère dans "Film", contrainte qui n'apparaît pas en HQL puisqu'elle est représentée par une association entre objets dans le graphe virtuel.

La définition du **mapping** permet à Hibernate d'engendrer le code SQL correct.

## Jointures (suite)

On peut adopter en HQL une expression plus proche de la logique SQL, mais moins naturelle du point de vue d'une logique "graphe d'objet".

---

```
select film  
from Film as film, Artiste as a  
where film.realisateur = a  
and a.nom='Eastwood'
```

---

**Le code SQL engendré est le même.**

Il n'y a donc aucune différence en termes de performance ou d'expressivité.

## Jointures (suite)

C'est une autre spécificité "objet" de HQL : on compare directement **l'identité** de deux objets (un artiste, un réalisateur) avec la clause :

---

```
where film.realisateur = a
```

---

et pas, comme en SQL, **l'égalité** des clés primaires de ces objets.

Mais il est possible d'effectuer directement une jointure sur les clés en HQL :

---

```
select film  
from Film as film, Artiste as a where  
film.realisateur.id = a.id and  
a.nom='Eastwood'
```

---

Ici, "id" est un mot-clé représentant la clé primaire, quel que soit son nom réel.

## La clause “select”

Autre exemple de jointure, avec une association un à plusieurs. On sélectionne tous les films dont un des rôles est **McClane** :

---

```
select distinct film
from Film as film
join film.roles as role where
role.nom= 'McClane'
```

---

La clause “select” prend ici une importance particulière : dans la mesure où nous accédons aux extensions de deux classes (“Film” et “Role”), il est préférable de préciser quels sont les objets que nous plaçons dans le résultat.

Sans clause “select”, on obtient une liste de paires d’objets (un film, un rôle), soit une instance de “List< Object[]>”.

Pour chaque élément de la liste, le premier composant du tableau est un “Film”, le second un “Rôle”.

Le film est dupliqué autant de fois qu’il a de rôles associés, ce qui rend indispensable l’utilisation du mot-clé “distinct”.

## La clause "select"

Exécutez par exemple la requête suivante.

---

```
select film.titre, role.nom from  
Film as film  
join film.roles as role
```

---

On obtient autant de fois le titre du film qu'il y a de rôles.

Un tel résultat est assez laborieux à traiter, et on perd en grande partie les avantages de HQL (à savoir instancier un résultat directement exploitable).

Dernier exemple, tous les films où Clint Eastwood joue un rôle :

---

```
select distinct film  
from Film as film  
join film.roles as role  
join role.pk.acteur as acteur  
where acteur.nom= 'Eastwood'
```

---

Notez le chemin "role.pk.acteur", exprimant un **chemin** dans le graphe des objets et composants. Je vous invite à consulter la requête SQL engendrée par Hibernate pour cette jointure.



# La clause “join” de HQL

Les jointures de HQL sont les mêmes que celles du SQL ANSI.

“[inner] join” : exprime une jointure standard pour impliquer (pas forcément matérialiser) les objets associés;

“left [outer] join” : exprime une **jointure externe à gauche** : si par exemple un film n’a pas de metteur en scène, il (le film) fera quand même partie du résultat, avec un objet “realisateur” associé à “null”;

“right [outer] join” est la complémentaire de “left join” (et a donc peu d’intérêt en pratique).

Pour bien voir la différence, exécutez les deux requêtes suivantes :

---

```
from Film as film  
inner join film.roles as role
```

---

et

---

```
from Film as film  
left outer join film.roles as role
```

---

La seconde devrait vous ramener **aussi** les films pour lesquels aucun rôle n’est connu. Cette requête renvoie une instance de “List< Object[]> “, qu’il faut ensuite décoder, pour obtenir, respectivement, des films et des rôles.

## La clause “join” de HQL

si, à partir de l'un des films sélectionnés, on veut accéder aux rôles par la collection “film.roles”, Hibernate a-t-il déjà instancié cette collection, ou va-t-il devoir à nouveau accéder à ces rôles avec une requête SQL ?

## La clause “join” de HQL

si, à partir de l'un des films sélectionnés, on veut accéder aux rôles par la collection “film.roles”, Hibernate a-t-il déjà instancié cette collection, ou va-t-il devoir à nouveau accéder à ces rôles avec une requête SQL ?

Réponse : une nouvelle requête sera effectuée pour trouver les rôles du film, ce qui est dommage car la jointure SQL sera effectuée deux fois.

**Les requêtes HQL que nous présentons ne créent pas de graphe, mais instancient simplement les objets sélectionnés, sans lien entre eux.**

Dit autrement : le fait d'effectuer une jointure entre un film et des rôles **n'implique pas** que le graphe **film-rôles** est matérialisé dans le cache de premier niveau.

## La clause “join” de HQL

Ce n'est pas si illogique que cela en a l'air, et cela rend possible d'appliquer des **restrictions** sur les objets sélectionnés, comme dans l'exemple ci-dessous.

---

```
from Film as film
left outer join film.roles as role where
role.nom='McClane'
```

---

Si on matérialisait le graphe à partir des objets sélectionnés, la collection “roles” de chaque film serait incomplète puisqu'elle ne comprendrait que ceux dont le nom est “McClane”.

En résumé, il faut bien distinguer deux motivations différentes, et partiellement incompatibles :

- la **sélection** d'objets satisfaisant des critères de restriction;
- la **matérialisation** d'une partie du sous-graphe de données.

Les requêtes HQL que nous présentons ici correspondent à la première motivation. Il est possible d'utiliser HQL **aussi** pour la matérialisation du graphe de données, avec une option “fetch” que nous étudierons dans le prochain chapitre.

# La clause “join” de HQL

- Ces deux motivations peuvent être **incompatibles** : si je **restreins** avec la clause **where** les objets sélectionnés (les films avec un rôle McClane), je ne peux pas obtenir en même temps un graphe complet (tous les rôles des films, dont un des rôles est McClane)

Pour l’instant, on se concentre sur l’utilisation de HQL pour sélectionner et matérialiser les objets qui vous intéressent, sans chercher à optimiser les performances ou éliminer les redondances de requêtes SQL engendrées.

Cela revient à **toujours utiliser** la clause “select” pour garder les objets qui nous intéressent, et à utiliser les jointures pour exprimer des restrictions.

Par exemple :

---

```
select distinct film
from Film as film
left outer join film.roles as role where
role.nom='McClane'
```

---

Comme en SQL, une autre manière d’exprimer une jointure est d’utiliser des sous-requêtes, qui sont présentées un peu plus loin.

## Table des pays

Voici la requête qui parcourt la table des pays, avec l'API *Criteria* :

---

```
public List<Pays> lecturePays() {  
    Criteria criteria = session.createCriteria(Pays.class);  
    return criteria.list();  
}
```

---

Plus facile de faire plus simple, on aurait même pu l'écrire en une seule ligne...

Bien sûr pour des requêtes plus complexes, la construction est plus longue.

Voici la méthode équivalente en HQL :

---

```
public List<Pays> lecturePaysHQL() {  
    Query query = session.createQuery("from Pays");  
    return query.list();  
}
```

---

C'est presque la même chose, mais on introduit une chaîne de caractères avec une expression qui n'est pas du java (et qui n'est donc pas contrôlée à la compilation).

# Visual Paradigm Academic Edition

Visual Paradigm Standard Edition est disponible pour les étudiants sur les postes windows du réseau ENSEIGNEMENT:

- génère depuis un diagramme de classe UML du code Java, C++, .Net (C#.NET, VB.NET et C++.NET), PHP 5.0, Python, Objective-C, et fait du reverse et de la synchronisation.

- génère depuis un diagramme de classe UML un schéma Entité-association étendu et les synchronise. A partir du diagramme Entité, association, VP génère le modèle logique puis les modèles physiques Oracle, MySQL, Microsoft SQL Server, HSQL, PostgreSQL, Derby, Firebird, SQLite. Fait du reverse Engineering.

- génère depuis un diagramme de classe UML un diagramme ORM et les synchronise. A partir du diagramme ORM, VP génère le code en Java pour Hibernate avec hibernate XML ou annotations JPA pour les bases de données cibles Oracle, MySQL, Microsoft SQL Server, HSQL, PostgreSQL, Derby, Firebird, SQLite. A partir du diagramme ORM, VP génère le code en PHP 5.0 pour Doctrine et EZPDO pour les bases de données cibles Oracle, MySQL, Microsoft SQL Server, PostgreSQL. A partir du diagramme ORM, VP génère le code en C# pour NHibernate pour les bases de données cibles Oracle, MySQL, Microsoft SQL Server, PostgreSQL, Firebird, SQLite.

Enfin VP est intégrable à Eclipse et Visual Studio.

# Classes persistantes dans Hibernate

- Garantie par Hibernate de l'équivalence de l'identité persistante (ou de base de données) et de l'identité Java mais uniquement à l'intérieur de la portée d'une session particulière
- Méthodes **equals()** et **hashCode()** à surcharger en particulier si :
  - Instances de classes persistantes gérées dans un Set (manière recommandée pour représenter des associations pluri-valuées)
  - Utilisation du réattachement d'instances détachées
  - Utilisation d'identificateur composite



# Classes persistantes dans Hibernate

## Règles pour surcharger `equals()` et `hashCode()`:

- cf . <http://www-128.ibm.com/developerworks/java/library/j-jtp05273.html>
- Recommandation Hibernate : implémenter **`equals()`** et **`hashCode()`** en utilisant *l'égalité par clé métier*
- Exemple de surcharge de **`equals()`** :

```
public boolean equals(Object other) {  
    // Test d'égalité de référence en mémoire  
    if (this == other) return true;  
    //Vérification de la classe dont other est instance  
    if ( !(other instanceof MyClass) ) return false;  
    // Test de comparaison des propriétés appartenant  
    // à la clé métier  
    final MyClass obj = (MyClass) other;  
    if ( !obj.getProp1().equals( getProp1() ) )  
        return false;  
    ...  
    return true; }  

```

## *hashCode()* et *equals()*

dans certains cas, il est recommandé d'implanter les méthodes *hashCode()* et *equals()*, dans les classes d'objets persistants

Quand ? :

- Si des instances d'une classe persistante doivent être conservées dans une collection (*Set*, *List* ou *Map*) qui couvre plusieurs sessions Hibernate, alors il est nécessaire de
  - fournir une implantation spécifique de *hashCode()* et *equals()*

Il vaut mieux éviter cette situation, car elle soulève des problèmes qui n'ont pas de solution entièrement satisfaisante.

On peut donc décider de toujours éviter de se mettre dans ce mauvais cas.

- Mais on va tâcher de bien comprendre ce qui se passe, et voir la fragilité introduite dans l'application par ce type de pratique.

## Le problème

---

```
// On maintient une liste des utilisateurs
Set<User> utilisateurs;

// Ouverture d'une première session
Session s1 = sessionFactory.openSession();

// On ajoute l'utilisateur 1 à la liste
User u1 = s1.load(User.class, 1); utilisateurs.add(u1);

// Fermeture de s1
s1.close();

// Idem, avec une session 2
Session s2 = sessionFactory.openSession();
User u2 = s2.load(User.class, 1);
utilisateurs.add(u2);
s2.close();
```

---

Les deux objets persistants *u1* et *u2*, correspondent à la même ligne,

- *Mais ils ne sont pas identiques* puisqu'ils ont été chargés par deux sessions différentes.

Ils sont insérés dans le *Set utilisateurs*, et comme leur *hashCode* (qui repose par défaut sur l'identité des objets) est différent, ce *Set* contient donc un doublon, ce qui ouvre la porte à toutes sortes de *bugs*.

Il faut dans ce cas implanter *hashCode()* et *equals()* non pas sur l'identité objet, mais en tenant compte de la *valeur* des objets pour détecter que ce sont les mêmes.

- le problème ne se pose pas si on reste dans le cadre d'une seule session.

## Solution

Voici un exemple d'implantation de ces deux méthodes, en supposant que le nom de l'utilisateur est une clé naturelle (ce qui est faux, bien sûr).

```

1      @Override
2      public int hashCode() {
3          HashCodeBuilder hcb = new HashCodeBuilder();
4          hcb.append(nom);
5          return hcb.toHashCode();
6      }
7
8      @Override
9      public boolean equals(Object obj) {
10         if (this == obj) {
11             return true;
12         }
13         if (!(obj instanceof User)) {
14             return false;
15         }
16         User autre = (User) obj;
17         EqualsBuilder eb = new EqualsBuilder();
18         eb.append(nom, autre.nom);
19         return eb.isEquals();
20     }

```

Il est donc souvent déconseillé d'ouvrir et fermer des sessions, à moins d'avoir une excellente raison et de comprendre l'impact

De même, évitez de stocker dans des structures annexes des objets persistants, la base de données est là pour ça.

# Quelques patterns pour la persistance des objets avec DAO

- Modèle DTO (pour EJB 1 et 2)
- Modèle DAO
- Problèmes avancés avec les DAOs
- Modèle fabrique abstraite

Il est plus fréquent de changer la façon d'effectuer la persistance que de changer le modèle « métier »

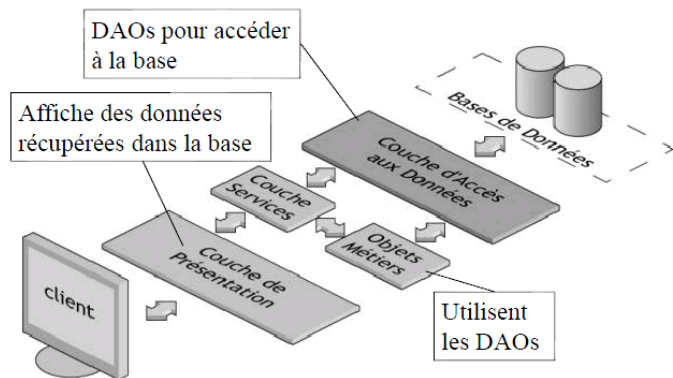
Pour faciliter les changements dans la persistance faut isoler le plus possible le code qui gère la persistance.

La persistance est isolée dans des objets spécifiques, les DAO (*Data Access Objects*)

Sans doute le modèle de conception le plus utilisé dans le monde de la persistance

# Utilisation des DTOs avec les DAOs

- Les DAOs sont situés dans une couche proche de la base de données
- Le code qui utilise les DAOs est souvent situé sur une autre couche distante
- Les DTOs (synonyme : *Transfert Object*, *TO*) peuvent être utilisés pour transporter les données entre les DAOs et cette autre couche (ou plus généralement entre 2 couches distantes)
- Le client (celui qui veut les données pour faire le traitement) demande un objet qui contient toutes les valeurs dont il a besoin
- Cet objet, un *Data Transfert Object (DTO)*, est construit sur le site distant et passé en une seule fois au client
- Un DTO contient l'état d'un objet (ou de plusieurs), mais pas son comportement ; il est donc facilement transportable sur le réseau (il est sérialisé ; voir cours sur les entrées-sorties)





# DAO (Data Access Object)

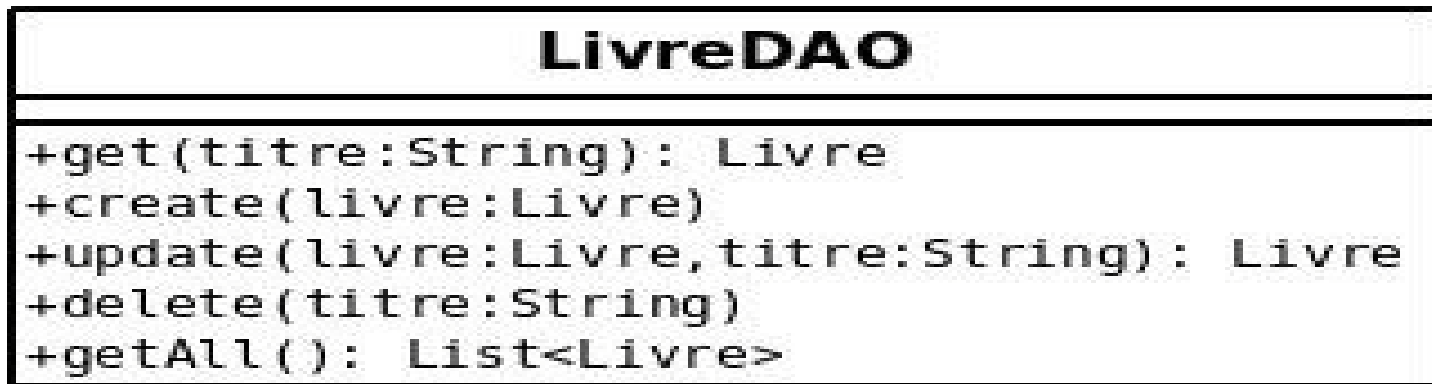
- L'acronyme CRUD désigne les opérations de base de la persistance : *create, retrieve, update et delete*
- Ces 4 opérations de base sont implémentées dans un DAO.
- Pour neutraliser la couche de persistance :
  - Il faut cacher le type de connexion .
  - Il faut permettre de gérer les transactions
  - Il faut des exceptions spécifiques
  - Une fabrique abstraite



# Le patron de conception DAO

DAO (Data Access Object): patron de conception

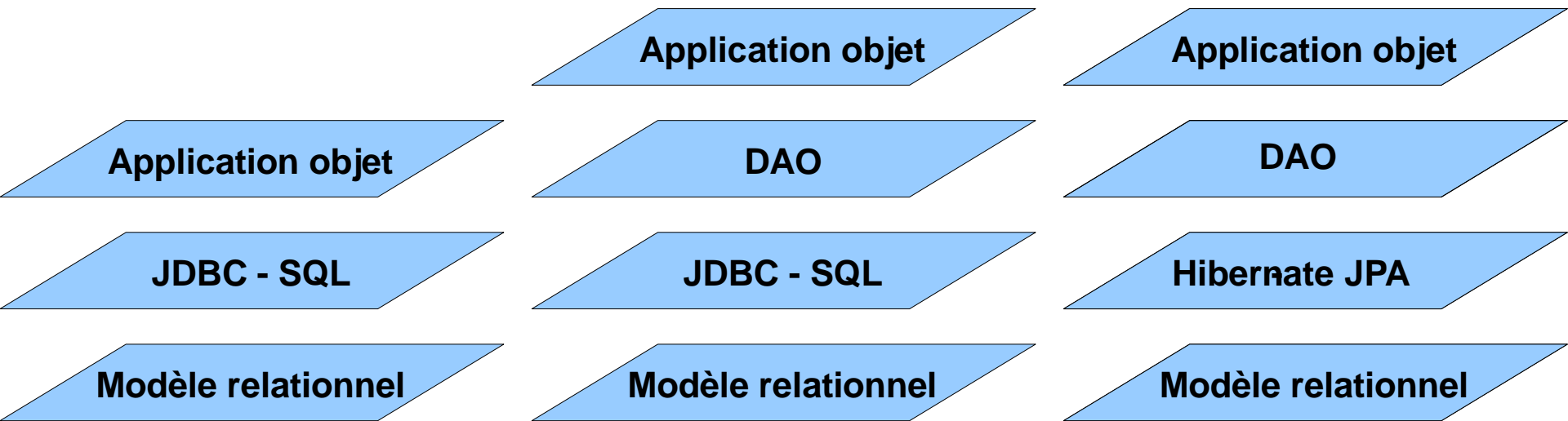
- pour définir une couche d'accès au données (CRUD)
- Classe metier Livre → DAO  
LivreDAO.





# Les ORM ( Object Relational Mapping ) : des cadres applicatifs pour faciliter le mapping objet/relationnel

3 modèles architecturaux: sans DAO, avec DAO, avec DAO + ORM



# Le framework Spring

- Spring s'appuie principalement sur l'intégration de trois concepts clés :
  - l'[inversion de contrôle](#) ou injection de dépendance ;
  - la [programmation orientée aspect](#) ;
  - une couche d'abstraction.
- La couche d'abstraction permet d'intégrer d'autres [frameworks](#) et [bibliothèques](#) avec une plus grande facilité. Cela se fait par l'apport ou non de couches d'abstraction spécifique à des frameworks particuliers. Il est ainsi possible d'intégrer un module d'envoi de mails en toute facilité.
- Ce framework, grâce à sa couche d'abstraction, ne concurrence pas d'autres frameworks dans une couche spécifique d'un modèle architectural [MVC](#) mais s'avère un framework multi-couches pouvant s'insérer au niveau de toutes les couches ; modèle, vue et contrôleur. Ainsi il permet d'intégrer [Hibernate](#) ou [iBATIS](#) pour la couche de persistance ou encore [Struts](#) pour la couche présentation.

# Inversion de Contrôle

- IoC se rapporte aux dépendances entre objets.
- L'approche traditionnelle ("Pull") :
  - Une instanciation directe
  - On demande à une fabrique pour l'implémentation
  - Recherche d'un service via JNDI
- L'approche "Push" :
- Quelque chose externe aux objets pousse leurs dépendances en leur sein. L'objet ignore comment il les obtient. Il lui suffit de considérer qu'elles existent.
- L'approche "Push" est appelée "l'injection de dépendances".

# Exemple Pull

```
public class BookDemoServicePullImpl implements BookDemoService {  
  
    public void addPublisherToBook(Book book) {  
  
        BookDemoFactory factory = BookDemoFactory.getFactory();  
        BookDemoDao dao = factory.getBookDemoDao();  
  
        String isbn = book.getIsbn();  
        if (book.getPublisher() == null && isbn != null) {  
            Publisher publisher = dao.findPublisherByIsbn(isbn);  
            book.setPublisher(publisher);  
        }  
    }  
}
```

# Exemple Push (Injection de dépendences )

```
public class BookDemoServiceImpl implements BookDemoService {  
  
    private BookDemoDao dao;  
  
    public void addPublisherToBook(Book book) {  
  
        String isbn = book.getIsbn();  
        if (book.getPublisher() == null && isbn != null) {  
            Publisher publisher = dao.findPublisherByIsbn(isbn);  
            book.setPublisher(publisher);  
        }  
  
    }  
  
    public void setBookDemoDao(BookDemoDao dao) {  
        this.dao = dao;  
    }  
  
}
```

# Coupler un composant à un autre avec Ioc

- IoC (Inversion of Control): Pattern permettant de fournir aux POJOs (Plain Old Java Object = un objet Java simple et classique) les objets qu'ils vont utiliser.
- Le principe consiste à inverser le contrôle avec une classe supplémentaire chargée d'initialiser les dépendances entre les différents composants.
- On ajoute une indirection et on sort ainsi des composants la responsabilité de créer les classes dépendantes.
- La classe `Initializer` réalise l'inversion :

```
myobj = new MyBusinessObject() ;  
BookDemoService bds = new BookDemoServiceImpl() ;  
myobj.setBookDemoDao(bds) ;
```

# Spring

- L'injection de dépendances consiste à injecter des dépendances dans un objet.
- L'inversion de contrôle est un style utilisant l'injection de dépendances pour cabler ensemble les couches applicatives.
  - Il permet une faible imbrication des couches applicatives car les dépendances ne sont pas codées en dur mais ce trouve dans un fichier de configuration.
  - Les objets sont moins tributaires de leur environnement, ce qui les rend testables plus facilement.
- Spring est un conteneur d'inversion de contrôle qui gère pour vous cette dernière. Une application est un ensemble d'objets que Spring appelle des beans, parce qu'ils suivent la norme JavaBean de nommage des accesseurs et initialiseurs (getters/setters) des champs privés d'un objet. Les objets qui, dans une application, ont pour rôle de rendre un service sont souvent créés en un seul exemplaire. On les appelle des singletons.
- Les quatre informations nécessaires au constructeur de cet objet sont définies à l'intérieur d'une balise `<bean> . . . </bean>`. On aura autant de telles balises `<bean>` que de singletons à construire.
- La solution avec Spring va éliminer le besoin qu'a la classe métier de connaître le nom de la classe d'accès aux données dont elle a besoin. Cela permettra d'en changer sans toucher au code java de la classe métier. Spring va permettre de créer en même temps les deux singletons, celui de la couche d'accès aux données et celui de la couche métier

# Exemple de fichier Spring applicationContext.xml

```
<beans>
```

```
    <bean id="bookDemoDao"
```

```
        class="com.bookdemo.dao.hibernate.BookDemoDaoHibernateImpl">
```

```
        <property name="sessionFactory">
```

```
            <ref local="sessionFactory"/>
```

```
        </property>
```

```
    </bean>
```

```
    <bean id="bookDemoService"
```

```
        class="com.bookdemo.service.impl.BookDemoServiceImpl">
```

```
        <property name="bookDemoDao">
```

```
            <ref bean="bookDemoDao"/>
```

```
        </property>
```

```
    </bean>
```

```
</beans>
```



# Lanceur/configuration d'une application web

- Une application web est initialisée et lancée par le conteneur web sur la base du fichier `web.xml`.
- Créer un contexte applicatif particulier au démarrage ne consiste qu'à une seule déclaration dans le fichier `web.xml` :

```
<listener>  
    <listener-class>  
        org.springframework.web.context.ContextLoaderListener  
    </listener-class>  
</listener>
```

- Il y a en plus le fichier `applicationContext.xml`.
- Mais aucune ligne de code java et aucunes des classes de l'application a une dépendance vis à vis de `spring.jar` !



## Integration dans Spring

# JDBC + Hibernate + Spring

Définition d'une source de donnée :

```
<bean id="dataSource"
  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName">
    <value>${hibernate.connection.driver_class}</value>
  </property>
  <property name="url">
    <value>${hibernate.connection.url}</value>
  </property>
  <property name="username">
    <value>${hibernate.connection.username}</value>
  </property>
  <property name="password">
    <value>${hibernate.connection.password}</value>
  </property>
  <property name="connectionProperties">
    <props>
      <prop key="hibernate.dialect">${hibernate.properties.dialect}</prop>
    </props>
  </property>
</bean>
```

[francois.jully@inp-toulouse.fr](mailto:francois.jully@inp-toulouse.fr)

48  
6

# Hibernate avancé : création et manipulation de schéma

- Création automatique du schéma
- Mise à jour du schéma (changement de structure)
- Facilités pour les migrations de données

```
<bean
  id="createSessionFactory"
  parent="abstractHibernateSessionFactory"
  lazy-init="true"
>
<description>
  This bean is used to create the database structures.
  Caution: leave inherited attribute lazy-init to true or the database will
  be re-created from scratch each time the application starts.
</description>
<property name="hibernateProperties">
  <props>
    <prop key="hibernate.hbm2ddl.auto">create</prop>
  </props>
</property>
</bean>
```

## Gestion des niveaux d'isolation

Hibernate s'appuie sur le niveau d'isolation fourni par le SGBD , et ne tente pas de ré-planter des protocoles de concurrence.

Le mode par défaut dans la plupart des systèmes est *read committed* ou *repeatable read*.

Ce niveau d'isolation (par défaut) n'offre pas toutes les garanties il est indispensable de se poser sérieusement la question des risques liés à la concurrence d'accès.

Dans une application transactionnelle, le mode *read uncommitted* ne devrait même pas être envisagé.

Il reste donc comme possibilités :

- d'accepter le mode par défaut, après évaluation des risques;
- ou de passer en mode *serializable*, en acceptant les conséquences (performances moindres, risques de *deadlock*);
- ou, enfin, d'introduire une dose de gestion "manuelle" de la concurrence en effectuant des verrouillages préventifs (dits, parfois, "verrouillage pessimiste").

## Gestion des niveaux d'isolation

La configuration Hibernate permet de spécifier le mode d'isolation choisi pour une application :

---

```
hibernate.connection.isolation = <val>
```

---

*val* est un des codes suivants :

- 1 pour *read uncommitted*
- 2 pour *read committed*
- 3 pour *repeatable read*
- 4 pour *serializable*

On peut donc spécifier un niveau d'isolation, qui s'applique à *toutes* les sessions, même celles qui ne présentent pas de risque transactionnel.

Dans ces conditions, choisir le niveau maximal (*serializable*) systématiquement représente une pénalité certaine.

Il semble préférable d'utiliser le niveau d'isolation par défaut du SGBD , et de changer le mode d'isolation à *serializable* ponctuellement pour les transactions sensibles.

## Niveaux d'isolation

Il ne semble malheureusement pas possible, avec Hibernate, d'affecter simplement le niveau d'isolation pour une session.

On en est donc réduit à passer par l'objet *Connexion* de JDBC.

---

```
1 session.doWork(  
2     new Work() {  
3         @Override  
4         public void execute(Connection connection) throws SQLException {  
5             connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);  
6         }  
7     });
```

---

*Le mode **serializable**, malgré ses inconvénients (apparition d'interblocages) est le plus sûr pour garantir l'apparition d'incohérences dans la base, dont la cause est très difficile à identifier.*

Une autre solution consiste à verrouiller explicitement, au moment de leur mise en cache, les objets que l'on va modifier.

## Verrouillage avec Hibernate

Hibernate se contente de transmettre les requêtes de verrouillage au SGBD

aucun verrou en mémoire n'est posé

(cela n'a pas de sens car chaque application ayant son propre cache, un verrou n'aurait aucun effet concurrentiel).

La principale utilité des verrous est de gérer le cas de la *lecture* d'une ligne/objet, suivie de *l'écriture* de cette ligne.

Par défaut, la lecture pose un verrou *partagé* qui n'empêche pas d'autres transactions de lire à leur tour.

## Verrouillage avec Hibernate

Le principe du verrouillage explicite est donc d'effectuer une lecture qui *anticipe* l'écriture qui va suivre.

C'est l'effet de la clause *for update*.

---

```
select * from xxx where .... for update
```

---

Le *for update* déclare que les lignes sélectionnées vont être modifiées ensuite.

Pour éviter qu'une autre transaction n'accède à la ligne entre la lecture et l'écriture, le système va alors poser un *verrou exclusif*.

Le risque de mise à jour perdue disparaît.



## Verrouillage avec Hibernate

Avec Hibernate, l'équivalent du *for update* est la pose d'un verrou au moment de la lecture.

---

```
Transaction tx = session.beginTransaction();
Film film = session.get(Film.class, filmId, LockMode.UPGRADE);
film.titre = ... ; // Mises à jour
tx.commit();
```

---

## Verrouillage avec Hibernate

L'énumération *LockMode* implique l'envoi d'une requête au SGBD avec une demande de verrouillage. Les valeurs possibles sont :

*LockMode.NONE*. Pas d'accès à la base pour verrouiller (mode par défaut).

*LockMode.READ*. Lecture pour vérifier que l'objet en cache est synchronisé avec la base.

*LockMode.UPGRADE*. Pose d'un verrou exclusif sur la ligne.

*LockMode.WRITE*. Utilisé en interne.

Gérer la concurrence dans le code d'une application est une opération lourde et peu fiable.

Se limiter au principe simple suivant : quand on lit une ligne que l'on va modifier ensuite, on place un verrou avec *LockMode.UPGRADE*.

Pour tous les autres cas, n'utilisez pas les autres modes et laissez le SGBD appliquer son protocole de concurrence.

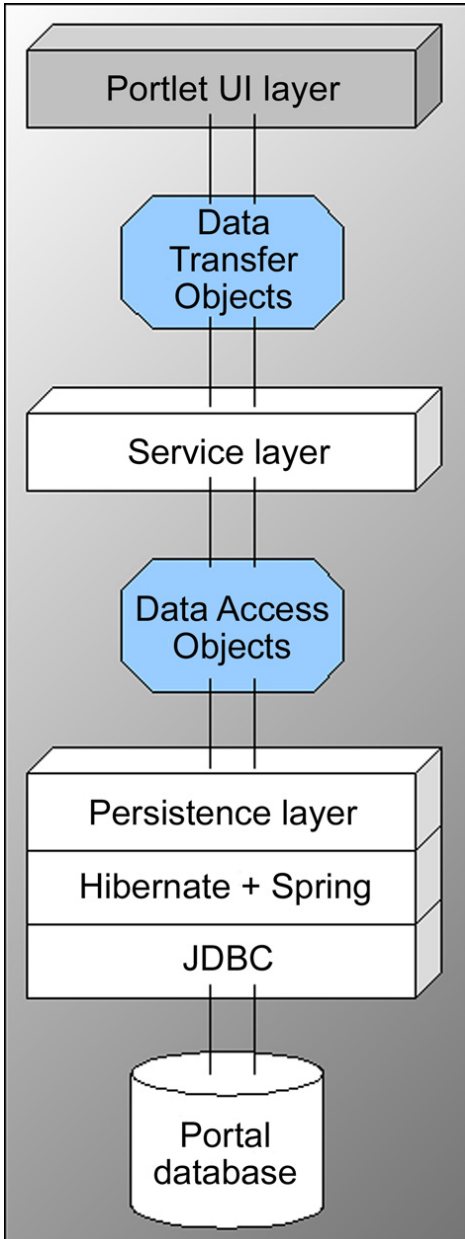
# Hibernate : modules

Différents modules :

- ***Hibernate Core*** : API native implémentant les services de base pour la persistance
  - Méta-données au format XML (+ annotations depuis la version 3.6)
  - Langage HQL et interface pour écrire des requêtes
- ***Hibernate Annotations*** (inclus dans *Hibernate Core* 3.6) : Remplacement des fichiers XML par des annotations JDK 5.0 implémentant les annotations du standard JPA + annotations spécifiques à Hibernate
- ***Hibernate Entity Manager*** : Implémentation de la partie des spécifications JPA concernant
  - Les interfaces de programmation,
  - Les règles de cycle de vie des objets persistants
  - Les fonctionnalités d'interrogation

*Hibernate Entity Manager* = *wrapper* au dessus du noyau Hibernate implémentant une solution complète de persistance JPA (cf. documentation Hibernate)

# ORM avec Liferay



- Le service Builder est un outil de génération de sources créé par Liferay.
- Basé sur Spring IoC et Hibernate, il permet de créer rapidement le socle d'une portlet en générant un ensemble de models, les scripts SQL associés à ceux-ci pour votre base de données, ainsi que les classes et interfaces des couches de persistance, de services et utilitaires nécessaires à la manipulation des entités souhaitées. Le service Builder va également vous générer les configurations Spring nécessaires au bon fonctionnement de l'ensemble.
- les entités sont définies dans un fichier service.xml dans le dossier \docroot\WEB-INF

# JPA : les besoins

- Simplifier le modèle de persistance
- Modèle de persistance léger
- Support de modélisation complexe
  - héritage, polymorphisme
- ORM standard et efficace
  - optimisé pour les SGBD relationnels
  - annotations standardisées
- Requêtes
- Plusieurs supports de persistance
  - notion de persistence unit (persistence.xml)

# JPA : Comportement

- Persistance des entités associées
  - lorsqu'un objet o est rendu persistant, les objets référencés par o devraient être rendus persistants
  - persistance par transitivité
  - comportement logique, mais...
- complexe !
  - cohérence
  - par exemple, que se passe-t-il si un objet supprimé est référencé par un autre objet ?
- JPA n'effectue pas de persistance par transitivité par défaut
  - plus de souplesse
  - possibilité d'avoir la persistance par transitivité avec cascade

# JPA : Comportement (cont.)

- Attribut cascade
  - permet de propager les effets d'une opération sur les entités associées
  - valeurs possibles: PERSIST, REMOVE, REFRESH, MERGE, ALL
  - `@OneToMany(cascade=CascadeType.PERSIST)`
- Par défaut, aucune opération n'est appliquée transitivement
- En JPA 2.0, suppression automatique des orphelins
  - `@OneToMany(cascade=CascadeType.PERSIST, orphanRemoval=true)`

# JPA : Comportement (cont.)

- Récupération des entités associées
  - lorsqu'une entité est récupérée (requête), est-ce qu'on récupère aussi les entités associées ?
  - et si on récupère les entités associées, est-ce que l'on récupère les entités associées des entités associées ?
  - et si... !
- JPA laisse le choix de récupérer ou non les entités associées
  - on peut choisir le mode de récupération: LAZY ou EAGER
    - EAGER: immédiat
    - LAZY: à la demande



# JPA : Comportement (cont.)

- LAZY
  - pour les associations avec des hiérarchies importantes
  - dans ce cas, l'entité associée n'est pas récupérée immédiatement
  - JPA remplace l'entité par un proxy qui permettra de récupérer l'entité plus tard si besoin
  - le proxy contient la clé primaire
- Par défaut
  - JPA est en mode EAGER pour 1-1 et N-1
  - JPA est en mode LAZY pour 1-N et M-N



# JPA : Interfaces principales

- EntityManagerFactory
  - utilisée pour créer des entity managers
  - une seule entity manager factory par persistence unit
- EntityManager
  - utilisé pour gérer le contexte de persistance
    - écriture/lecture des entités dans/de la base de données
  - opérations: persist, remove, find, refresh, createQuery,...
- Query, TypedQuery
  - utilisées pour la configuration des requêtes, la définition des paramètres et l'exécution des requêtes



```
public static void main(String[] args) {
```

## RESOURCE\_LOCAL avec Java SE

```
//Création de l'EntityManagerFactory
```

```
EntityManagerFactory emf =
```

```
    Persistence.createEntityManagerFactory("MyUnitName");
```

```
//Création de l'EntityManager
```

```
EntityManager em = emf.createEntityManager();
```

```
//Récupération d'une transaction
```

```
EntityTransaction tx = em.getTransaction();
```

```
//Début des modifications
```

```
try{
```

```
    tx.begin();
```

```
    em.persist(entity);
```

```
    tx.commit();
```

```
} catch (Exception e) {
```

```
    ...
```

```
    tx.rollback();
```

```
} finally {
```

```
    em.close();
```

```
    emf.close();
```

```
} Michel Dubois
```

```
}
```



```
public class PersonneEJB {
```

## RESOURCE\_LOCAL avec Java EE

```
//Récupération de l'EntityManagerFactory du conteneur JEE
@PersistenceUnit(unitName=" unit2")
private EntityManagerFactory factory;

public void methode() throws Exception {
    //Création de l'Entity Manager
    EntityManager entityManager = factory.createEntityManager();
    //Récupération de la transaction
    EntityTransaction entityTransaction =
entityManager.getTransaction();
    //Début de la transaction
    entityTransaction.begin();
    //Récupération d'une entité Personne
    Personne pers = entityManager.find(Personne.class, 12345);
    //Modification de son nom
    pers.setNom("nouveauNom");
    //Validation de la transaction
    entityTransaction.commit();
}
```

```
... Michel Dubois
```

```
}
```



## JTA avec Java EE

```
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class PersonneEJB {

    //Récupération d'un EntityManager via le conteneur JEE
    @PersistenceContext(unitName = "unitJTA")
    private EntityManager entityManager;

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void methode() throws Exception {

        Personne pers = entityManager.find(Personne.class, 12345);
        pers.setNom("nouveauNom");
    }
}
```

# ORM Implementations

## Java

- Hibernate (JBoss)
- TopLink (Oracle) → EclipseLink
- OpenJPA (Apache), .....

## .NET

- ADO.NET Entity Framework
- NHibernate ,...

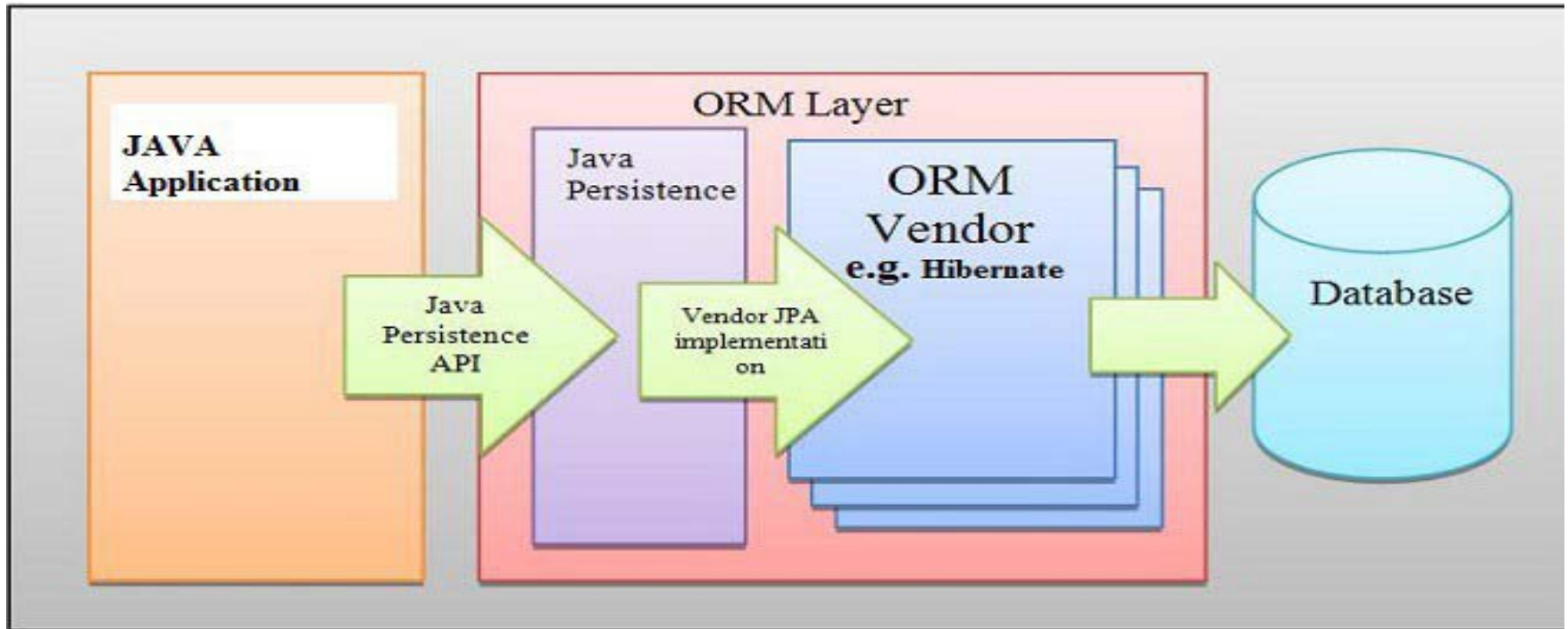
## PHP

- Doctrine, Redbean, ...

*... Every respectable programming platform has it*

# What JPA-ORM Should I Use?

It doesn't (shouldn't) matter from the programmer perspective...




You don't like Hibernate runtime performance anymore?  
Simply replace its libraries with another implementation,  
like EclipseLink or OpenJPA for instance.

# Autres solutions ORM

- **Normes Java :**
  - **EJB (*Entreprise Java Beans*) :**
    - Gestion de la persistance par conteneur (CMP- *Container-Managed Persistence* et BMP – *Beans Managed Persistence*)
    - Spécifications EJB3.0 (JSR 220 Mai 2006)
  - **JDO (Java Data Object) :**
    - Spécification de Sun 1999 – JDO 2.0 (JSR243 Mars 2006)
    - Abstraction du support de stockage
    - Implémentation libre : JPOX
  - **JPA (Java Persistence API) :** *Partie des spécifications EJB 3.0 (JSR 220 en Mai 2006 – JSR 316 en cours) concernant la persistance des composants*
- **Implémentation de JPA :**
  - **Hibernate (JBoss) : Solution libre faisant partie du serveur d'appli. JBoss – version 3.3**
    - implémentant les spécifications JSR 220 – complète et bien documentée - plugin Eclipse -
    - Gavin King (fondateur) membre de groupes d'expert d'EJB3
  - **TopLink (Oracle) : Solution propriétaire utilisée par la serveur d'application d'Oracle -**
    - **TopLink Essentials : version libre disponible dans Netbeans 5.5 ou le serveur d'application**  
(Java EE 5) *Glassfish de Sun, intégrée dans le projet EclipseLink (version 1 07/2008)*
  - **OpenJPA (Apache), Cayenne (Apache) ...**
- **"Comparaison " des solutions ORM :**
  - [http://se.ethz.ch/projects/shinji\\_takasaka/master\\_thesis\\_shinji\\_takasaka.pdf](http://se.ethz.ch/projects/shinji_takasaka/master_thesis_shinji_takasaka.pdf)
  - <http://terrazadearavaca.blogspot.com/2008/12/jpa-implementations-comparison.html>
  - <http://java.dzone.com/news/hibernate-best-choice?page=2>





# JP Query Language (JPQL)

- **Requêtes sur les Entités** et leur état persistant.
- **Des requêtes portables** et indépendant du datastore sous-jacent.
- Avec une syntaxe **SQL-like**

```
QL_statement ::= select_clause from_clause  
               [where_clause]  
               [groupby_clause]  
               [having_clause]  
               [orderby_clause]
```

```
SELECT p FROM Person p WHERE p.name LIKE "%Duke%"
```

# Requêtes JPQL

- Requêtes statiques
  - @NamedQuery, @NamedNativeQuery
- Requêtes dynamiques
  - définition de la requête à l'exécution
- EntityManager permet de créer et exécuter les requêtes
  - createNamedQuery, createQuery, createNativeQuery

# Recherche par identifiant

- `find` et `getReference` permettent de retrouver une entité en donnant son Id
- Elles ont deux paramètres
  - `Class<T>` pour indiquer le type de l'entité recherchée
  - `Object` pour indiquer la clé primaire

```
public void supprimeCommande(Long commandeId) {  
    Commande cmde = entityManager.find(Commande.class,  
    commandeId);  
    entityManager.remove(cmde);  
}
```

Michel Dubois



# Création de requêtes

## Dynamic Queries

```
public List<Person> findByName(String name)
{
    Query query = entityManager.createQuery(
        "SELECT p FROM Person p WHERE p.name LIKE :personName")
        .setParameter("personName", name)
        .setMaxResults(10);
    return
}    query.getResultList();
```

## Static Queries

```
@NamedQuery(name="findById",
    query="SELECT p FROM Person p WHERE p.id = :personId")
)
@Entity
@Table(name="persons")
class Person { ... }
```

Using a static query:

```
Person p = em.createNamedQuery("findById")
    .setParameter("personId", 1)
    .getSingleResult();
```

# Requêtes statiques

- Définition

```
@NamedQuery(name="clientParCodePostal",  
query = "SELECT c FROM Client c WHERE  
c.adresse.codePostal = :cp")  
@Entity public class Client {  
  
...  
}
```

- Utilisation

```
// Utilisation  
public List rechercheClientParCodePostal(int cp) {  
return em.createNamedQuery("clientCodePostal")  
.setParameter("cp", cp)  
.setMaxResults(20)  
.get  
}
```

Michel Dubois

# Requêtes statiques - Paramétrage

```
@NamedQuery(  
name="clientParNom",  
queryString="SELECT c FROM Client c " +  
"WHERE c.nom LIKE :nomClient"  
)
```

```
@PersistenceContext public EntityManager em;  
  
List clients =  
em.createNamedQuery("clientParNom")  
.setParameter("nomClient", "Marchand")  
.getResultList();
```

# Requêtes dynamiques

```
public List rechercheAvecNom  
(String nom) {  
    return em.createQuery (  
        "SELECT c FROM Client c " +  
        "WHERE c.nom LIKE :nomClient")  
        .setParameter("nomClient", nom)  
        .setMaxResults(10)  
        .getResultList();  
}
```

# Polymorphisme

- Toutes les requêtes sont polymorphes
- Un nom de classe dans la partie FROM désigne aussi bien l'entité désignée que ses sous-classes !
- Exemple
  - `select count(a) from Articles as a`
  - dénombre le nombre d'instances de la classe Articles et de toutes les sous-classes de Articles



# Expressions de chemins

- On peut utiliser ces expressions pour naviguer entre les entités
- Si `e` est un alias pour `Employe`
  - `e.departement` désigne le département d'un employé
  - `e.projets` désigne la collection de projets auxquels participe l'employé
- On peut chaîner les navigations à condition que la navigation précédente ne renvoie pas un ensemble
  - `e.departement.nom` désigne le nom du département de l'employé
  - `e.projets.nom` renvoie une erreur !

# Jointures JPQL

- Trois types
  - jointure interne
  - jointure externe
  - jointure avec récupération des données en mémoire (join fetch)
- Exemples
  - `select p.nom from Employe e join e.projets p`
  - `select d.nom from Employe e, Departement d where d=e.departement`
  - `select e.departement.nom from Employe e`
- Pas plus de 2 niveaux dans une expression de chemin pour une jointure
  - `select e,p from Employe e join e.participations.projet p` renvoie une exception
  - `select e,p from Employe e join e.participations pa join pa.projet p` est ok

# Exemples de navigation entre entité

Une expression peut traverser, ou naviguer entre entités reliées. JP-QL navigue entre des entités connexes, gérant les jointures SQL entre tables de manière automatique

## Utilisation de JOIN

```
- List<Person> persons = entityManager().createQuery(
    "select p from Person p join p.departaments as d "
    + "where d.name = :dept "
    + "order by p.name").
    setParameter("dept", someDeptName).
    getResultList();
```

## Utilisation de IN

```
- select distinct p from Person, in (p.departaments) d
```

*The **p variable** represents the Person entity, and the **d variable** represents the related Department entity. The declaration for d references the previously declared p variable. The **p.departaments** expression navigates from a Person to its related Departament.*

# Native Queries

Going back to good old SQL ...

## Simple SQL queries

```
BigInteger count = (BigInteger)
    entityManager.createNativeQuery( "select count(*) from
    persons where date_of_birth=:date"). setParameter("date",
    birthDate).
    getSingleResult();
```

## SQL queries mapped to entites

```
String sqlQuery = "select * from persons where date_of_birth = ?";
Query q = entityManager.createNativeQuery(sqlQuery, Person.class);
q.setParameter( 1, birthDate);
List<Person> persList = q.getResultList();
```

# Mise en œuvre à partir d'un client Java Standard Edition

```
public static void main(String[] args) {  
    EntityManagerFactory emf =  
        Persistence.  
            createEntityManagerFactory("EmployeeService");  
    EntityManager em = emf.createEntityManager();  
    Collection employees =  
        em.createQuery("SELECT e FROM Employee e").  
            getResultList();  
    ...  
}
```

# JPA Criteria API

The Context: is the query below valid?

```
String jpql = "select p from Person where p.age > 20";
```

Define **portable type-and-name safe** queries for entities

**Pure object oriented** queries

Define queries **dynamically** via construction of an *object-based CriteriaQuery* instance, rather than *string-based* approach used in JPQL

Very good performance (translated to native SQL queries)

# Example of Criteria Query

```
EntityManager em = ...  
//CriteriaBuilder is the factory for CriteriaQuery.  
CriteriaBuilder builder = em.getCriteriaBuilder();  
  
//The generic type argument declares the type of result  
//this CriteriaQuery will return upon execution  
CriteriaQuery<Person> query =  
builder.createQuery(Person.class);  
  
//Query expressions are the core units or nodes that are  
//assembled in a tree to specify a CriteriaQuery  
Root<Person> p = query.from(Person.class);  
Predicate condition = builder.gt(p.get(Person_.age), 20);  
query.where(condition);  
  
TypedQuery<Person> q = em.createQuery(query);  
List<Person> result = q.getResultList();
```

The equivalent JPQL query is

```
SELECT p FROM Person p WHERE p.age > 20
```

# Entity Manager

- C'est lui qui est chargé de gérer le cycle de vie des entités
- Équivalent de Hibernate Session ou JDO  
PersistenceManager
- 4 opérations principales
  - persist()
    - insère une entité dans la base
  - remove()
    - supprime une entité de la base
  - merge()
    - synchronise l'état des entités détachées
  - refresh()
    - recharge l'état des entités à partir de la base



# persist(), merge()

- **persist()**

```
public Commande creerCommande(Client c) {  
    Commande cmde = new Commande(c);  
    // Après l'appel à persist(), l'état de l'entité passe  
    // transient à managed.  
    // Au prochain commit, l'instance sera insérée dans la  
    // base de données  
    entityManager.persist(cmde);  
    return cmde;  
}
```

- **merge()**

- la méthode merge() retourne une copie gérée (managed) de l'entité (préalablement détachée)
- les modifications sur l'état persistant sont appliquées à l'instance gérée

# Agir sur le cycle de vie d'une entité

Une application peut être notifiée avant ou après chaque action sur une entité (chargement, persistance, mise à jour, suppression) grâce aux méthodes callback Annotations

- ▸ PrePersist/PostPersist
  - avant ou après que l'objet soit persistant
- ▸ PreUpdate/PostUpdate
  - • avant ou après la mise à jour des attributs de l'objet
- ▸ PreRemove/PostRemove
  - avant ou après la suppression de l'objet
- ▸ PostLoad
  - après que l'objet ait été chargé depuis la base de données

# Lifecycle Callbacks

*@PostLoad*

*@PrePersist, @PostPersist,*

*@PreUpdate, @PostUpdate*

*@PreRemove, @PostRemove*

```
public class PersonLogger {  
    @PostPersist  
    public void logAddition(Object p) {  
        getLog().debug ("Added:" + ((Person)p).getName());  
    }  
    @PreRemove  
    public void logDeletion(Object p) {  
        getLog().debug  
            ("Terminated:" + ((Person)p).getName());  
    }  
}  
  
@Entity  
@EntityListeners({ PersonLogger.class, ... })  
public class Person {  
    ...  
}
```

# Entity Manager et Persistence context

- Persistence context
  - représente un ensemble d'entités gérées à l'exécution
  - dire qu'une entité est gérée (dans l'état managed) signifie qu'elle appartient à un persistence context donné
  - les entités qui appartiennent à un persistence context ont un comportement commun
- Entity manager
  - il effectue les opérations du cycle de vie sur les entités
  - il gère un persistence context

# Entity Manager et Persistence Unit

- Une unité de persistance définit
  - toutes les entités
  - le mapping vers la base de données
- Un fichier persistence.xml définit une ou plusieurs unités de persistance

```
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">
<persistence-unit name="testPU" transaction-type="RESOURCE_LOCAL">
<provider>
oracle.toplink.essentials.ejb.cmp3.EntityManagerFactoryProvider
</provider>
<class>s3.Client</class>
<properties>
<property name="toplink.jdbc.url" value="jdbc:mysql://localhost:3306/testDB"/>
<property name="toplink.jdbc.user" value="root"/>
<property name="toplink.jdbc.driver" value="com.mysql.jdbc.Driver"/>
<property name="toplink.jdbc.password" value=""/>
<property name="toplink.target-database" value="MySQL5"/>
<property name="toplink.ddl-generation" value="create-tables"/>
</properties>
</persistence-unit>
</persistence>
```

# Exemple 2

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_1.xsd">
<persistence-unit name="HorensPU" transaction-type="RESOURCE_LOCAL">
<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
<class>ubs.horens.orm.Composante</class>
<class>ubs.horens.orm.Departement</class>
<class>ubs.horens.orm.Enseignant</class>
<class>ubs.horens.orm.Etablissement</class>
<class>ubs.horens.orm.PermissionGroupe</class>
<class>ubs.horens.orm.Permission</class>
<class>ubs.horens.orm.ProfilUtilisateur</class>
<class>ubs.horens.orm.Utilisateur</class>
<exclude-unlisted-classes>false</exclude-unlisted-classes>
<properties>
<!-- <property name="eclipselink.logging.level" value="SEVERE"/> -->
<!-- <property name="eclipselink.logging.file" value="eclipselink-jpa.log"/> -->
<!-- <property name="eclipselink.logging.logger" value="JavaLogger"/> -->
<property name="eclipselink.jdbc.driver" value="com.mysql.jdbc.Driver"/>
<property name="eclipselink.jdbc.url" value="jdbc:mysql://localhost:3306/horens?useUnicode=yes"/>
<property name="eclipselink.jdbc.user" value=""/>
<property name="eclipselink.jdbc.password" value=""/>
<property name="eclipselink.ddl-generation" value="none"/>
<property name="eclipselink.cache.shared.default" value="true"/>
</properties>
</persistence-unit>
</persistence>
```

# Exemple 3 :

## persistence.xml

```
<persistence>
  <persistence-unit name="MyApplicationPU"
    transaction-type="RESOURCE_LOCAL">

    <provider>org.hibernate.ejb.HibernatePersistence</provider>

    <class>myapp.entity.Person</class>
    <class>myapp.entity.Deparment</class>

    <properties>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.PostgreSQLDialect"/>
      <property > name="hibernate.connection.driver_class"
        value="org.postgresql.Driver"/>
      <property name="hibernate.connection.url"
        value="jdbc:postgresql://localhost/timetable"/>
    </properties>

  </persistence-unit>
</persistence>
```

# Types de transactions

- Il existe 2 types de transactions
  - transactions locales à une ressource
    - fournies par JDBC
    - associées à une seule base de données
  - transactions JTA
    - plus de fonctionnalités que les transactions JDBC
    - elles peuvent travailler avec plusieurs bases de données
- Dans JavaStandard Edition, transactions locales



# Création d'un EntityManager

## Container-Managed Entity Managers

```
@PersistenceContext  
EntityManager em;  
//  
@PersistenceUnit  
EntityManagerFactory emf;
```

```
<persistence-unit  
  name="MyApplicationPU"  
  transaction-type="JTA">
```

*The @PersistenceContext annotation can be used on any CDI bean, EJB, Servlet, Servlet Listener, Servlet Filter, or JSF ManagedBean.*

## Application-Managed Entity Managers

```
EntityManagerFactory factory =  
    Persistence.createEntityManagerFactory(  
        "MyApplicationPU", properties);  
EntityManager em = factory.createEntityManager();  
...  
em.close();  
factory.close();
```

```
<persistence-unit  
  name="MyApplicationPU"  
  transaction-type="RESOURCE_LOCAL">
```

# Entity transaction

- Si on n'utilise pas un serveur d'applications (par ex. Glassfish), on doit utiliser l'interface `javax.persistence.EntityTransaction`
- Une instance de `EntityTransaction` s'obtient par appel de la méthode `getTransaction()` de `EntityManager`

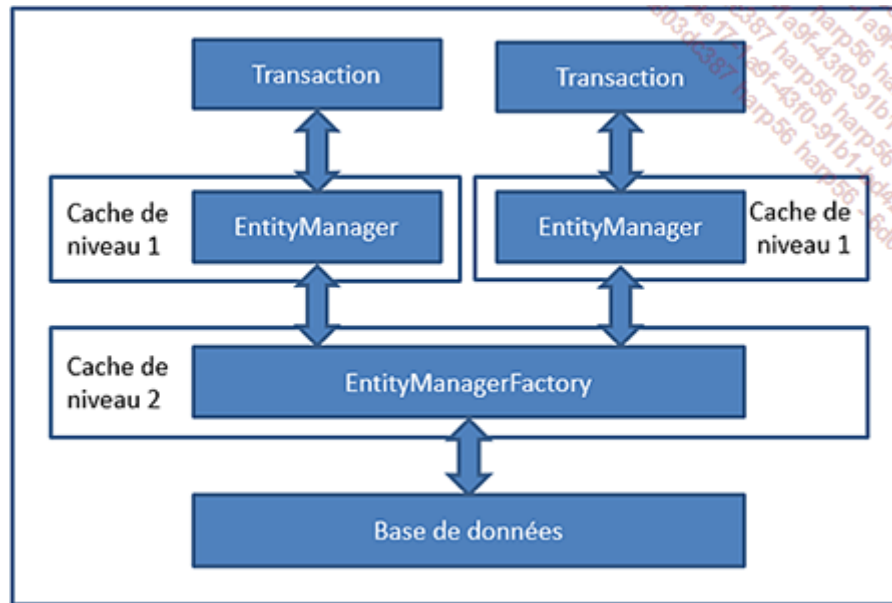
```
public interface EntityTransaction { EntityManager em;
public void begin();
public void commit();
public void rollback();
public void setRollbackOnly();
public boolean getRollbackOnly();
public void isActive();
}

...
try{
em.getTransaction().begin()
...
em.getTransaction().commit()
;
} finally{
em.close();
}
```

# Rollback/commit

- En cas de rollback
  - rollback dans la base de données
  - le contexte de persistance est vidé, i.e. toutes les entités sont détachées
  - les instances d'entités Java gardent les valeurs qu'elles avaient au moment du rollback
  - le plus souvent, il faut relancer les requêtes pour éviter d'avoir des entités avec des valeurs incorrectes
- En cas de commit
  - les modifications effectuées sur les entités gérées sont enregistrées dans la base de données
  - les modifications sont enregistrées dans la base, même si elles ont été effectuées avant le début de la transaction

# Les différents niveaux de caches



Une explication simplifiée de leur mode de fonctionnement est que, lorsque l'application veut récupérer des données, l'ORM vérifie dans un premier temps dans son cache, et s'il ne trouve pas les informations, il interroge la base de données. Les modifications sont faites au fur et à mesure dans le cache et transmises à la base de données lors du commit de la transaction.



# Chapitre 11

## Niveau externe

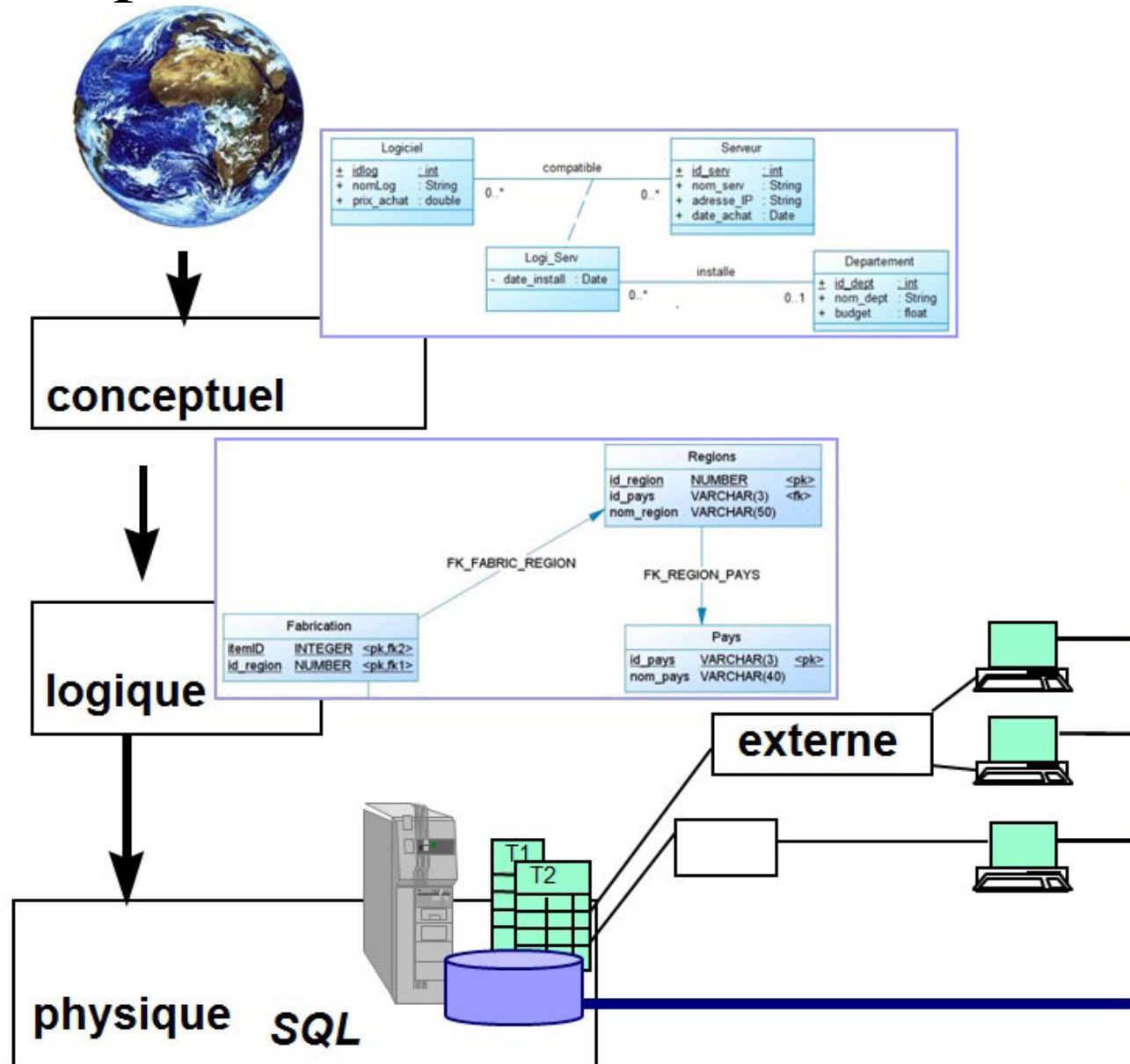
Michel Dubois

I.U.T. de Vannes

*Michel.Dubois@univ-ubs.fr*

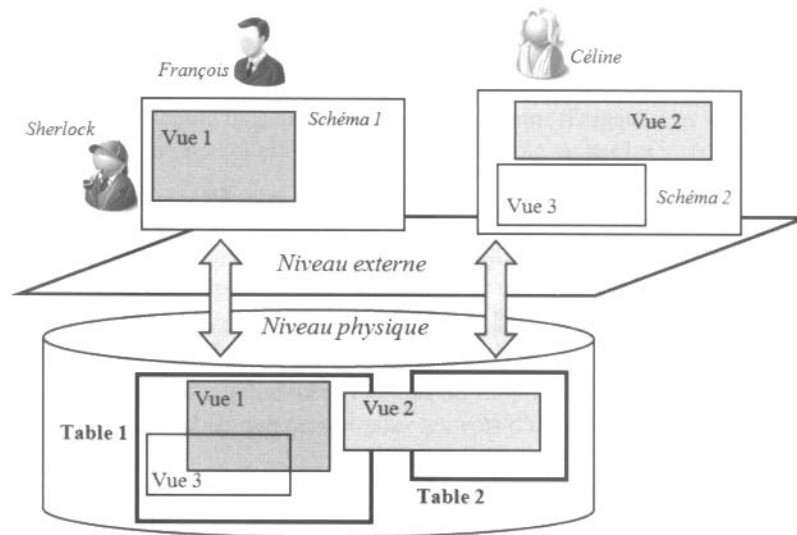
[Retour au plan](#)

# Niveau externe : dernière étape d'une implantation d'une base de donnée



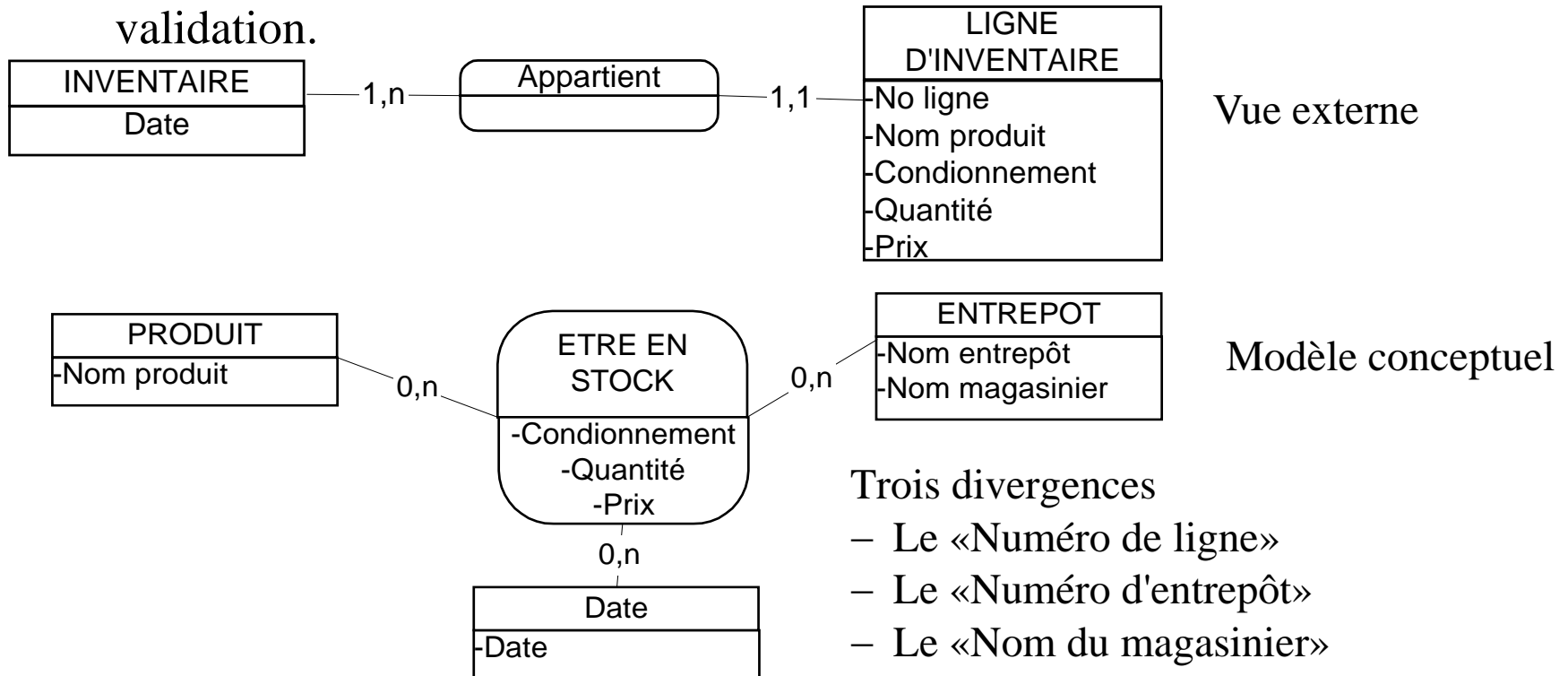
# Les vues relationnelles comme brique à l'indépendance logique

- Illustration de 3 accès à une base de données. Chaque utilisateur opère sur un schéma externe composé des vues accessibles pour son profil. Les données peuvent provenir de différentes tables et sont toujours stockées dans des tables.



# Merise et modèles externes

- Chaque acteur ayant une formation dans l'entreprise ne considère le S.I. que par rapport à ses préoccupations. Sa vue parcellaire du système est appelée modèle externe exprimés avec le formalisme MCD.
- Tous les points de vue sont pris en compte pour impliquer tous les acteurs dans une vision synthétique du système lors de la l'étape de validation.







# Définition du niveau externe

## = éviter l'accès direct aux tables

- Les vues relationnelles issues des tables SQL2 et les règles (PostgreSQL) et les déclencheurs instead of (Oracle) pour les mises à jour à travers elles.
- Routines SQL (procédures stockées, déclencheurs et UDF pour User Defined Function, paquetages)
- Les fonctions tables pour des vues paramétrables
- Les vues objets issues de tables SQL2 ou SQL3.

# VUES

Vue : table virtuelle calculée à partir d'autres tables grâce à une requête. **A chaque interrogation, elle est recalculée.** Elle n'est jamais obsolète. La clause ORDER est interdite par Oracle 8. Un effet de bord d'une clause GROUP BY est de trier les tuples. A partir d'Oracle 8i, les vues peuvent avoir la clause ORDER.

- Création

```
CREATE VIEW nom_vue AS requête  
[WITH CHECK OPTION];
```

```
CREATE VIEW Noms AS  
SELECT Nom, Prenom  
FROM Client;
```

- Destruction

```
DROP VIEW nom_vue
```

# *VUES AVEC DES ATTRIBUTS DERIVES*

Il faut attribuer un nom à un attribut dérivé de la requête

- Syntaxe :

```
CREATE VIEW nom_vue AS
SELECT Att1, Att1*2 [AS] NomAtt2, Att3
FROM .. WHERE..
;
```

- Autre syntaxe :

```
CREATE VIEW nom_vue(Att1, NomAtt2, Att3)
AS
SELECT Att1, Att1*2, Att3
FROM .. WHERE..
;
```

# *INTÉRÊT D'UNE VUE*

- Simplification de l'accès aux données en masquant les opérations de jointure
- Sauvegarde indirecte de requêtes complexes
- Calculs des attributs dérivés
- Résoudre des “traps” (problèmes avec regroupements)
- Présentation de mêmes données sous différentes formes adaptées aux différents usagers particuliers
- Support de l'indépendance logique
- Renforcement de la sécurité des données par masquage des lignes et des colonnes sensibles aux usagers non habilités

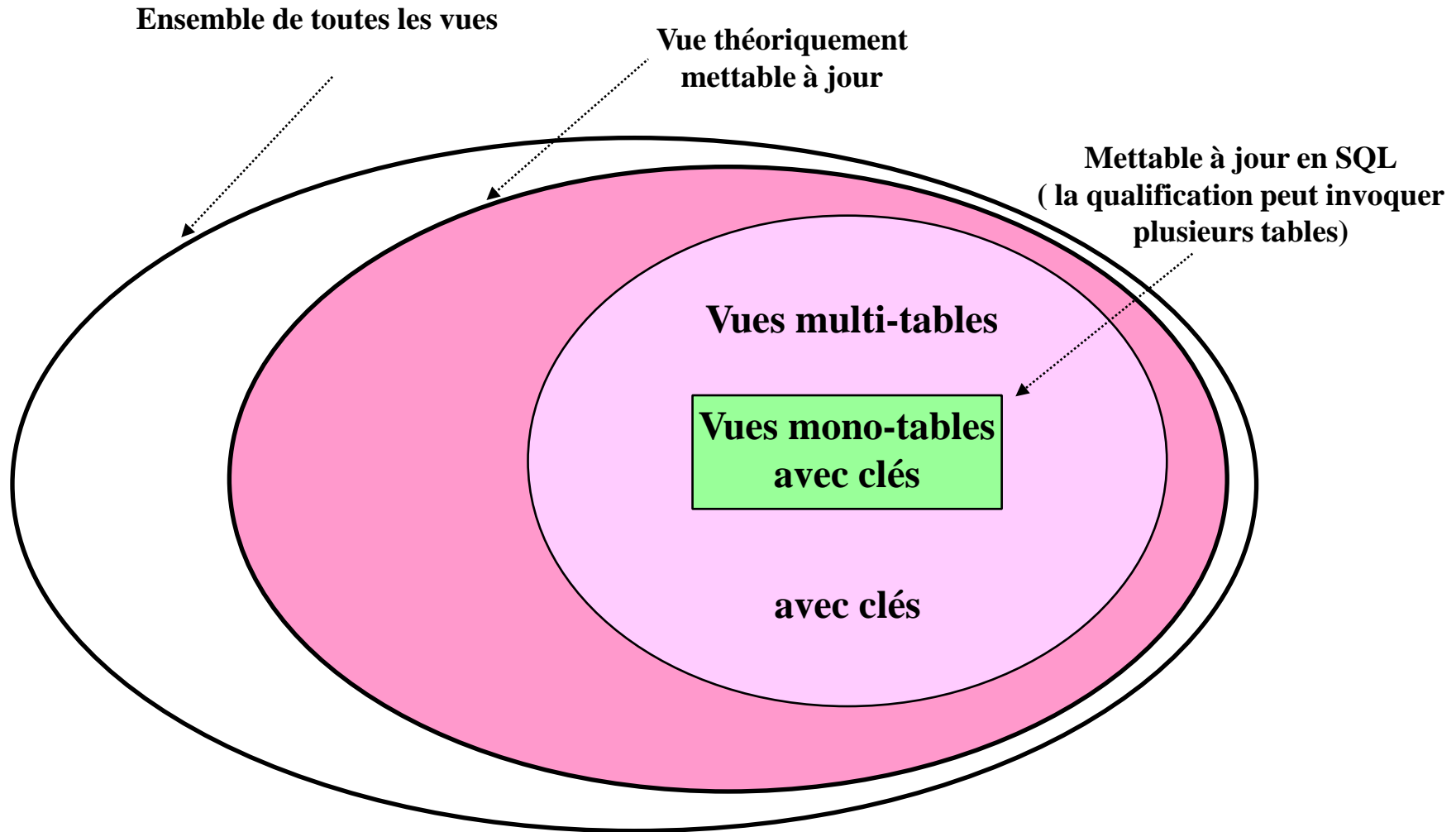
# MISE À JOUR À TRAVERS UNE VUE

- Vue modifiables (Updatable view)
  - Vue comportant suffisamment d'information pour permettre un report des mises à jour dans la base sans ambiguïté.
  - Les règles qui régissent les insertions ne sont pas identiques à celles qui concernent les mises à jour ou suppressions. Des vues complexes peuvent être modifiables, dans une certaine mesure, lorsqu'une table source est dite protégée par sa clé (*key preserved*). Une table est dite protégée par sa clé (*key preserved*) si sa clé primaire est préservée dans la clause de jointure et se retrouve en tant que colonne de la vue multitable (pouvant jouer le rôle de clé primaire de la vue). Pour être réellement préservée, cette clé doit être sans doublon dans la vue.
- Conditions suffisantes
  - Clés (primaire ou une candidate) des tables participantes déductibles de la vue
  - Vues mono-table avec clé de la table
  - Vues pas trop complexes (voir transparent suivant)

# *MISE À JOUR À TRAVERS UNE VUE*

- Le mot clé DISTINCT doit être absent.
- La clause FROM doit faire référence à une seule table. Par extension la vue doit être monotable concernant les jointures, pas de sous-requêtes, pas d'expressions de tables ou CTE, pas de CONNECT (Oracle)
- La clause SELECT doit faire référence directement aux attributs de la table concernée (pas d'attribut dérivé, pas de constantes, ni fonction de groupe, ni fonction de fenêtrage WINDOW), pas de ROWNUM ou ROW\_NUMBER.
- Les clauses GROUP BY, HAVING, ORDER BY sont interdites.

# CLASSIFICATION DES VUES



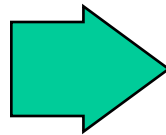


# *MODIFICATIONS PARADOXALES, INDÉSIRABLES*

- `CREATE VIEW v-nom [(colonne [, colonne]...)] AS sous-requête  
[ WITH CHECK OPTION ] ;`
- `CREATE VIEW bons_fournisseurs AS  
SELECT F, REMISE, VILLE  
FROM FOURNISSEUR WHERE REMISE > 15;`

Val. réelles

	1
	15




Val. virtuel.





## *CHECK OPTION*

- WITH CHECK OPTION protège contre les disparitions de tuples de vues:

```
UPDATE BONS_FOURNISSEURS  
SET REMISE = 5 WHERE F = 'F1' ;
```

serait alors rejeté.

De même l'ajout de tuples non concernés par la vue est bloqué.

```
INSERT INTO BONS_FOURNISSEURS(F, REMISE,  
VILLE) VALUES ('F10',10, 'PARIS')
```

## *CHECK OPTION*

- WITH CHECK OPTION protège contre les disparitions de tuples de vues:

UPDATE BONS\_FOURNISSEURS  
SET REMISE = 5 WHERE F = 'F1' ;

serait alors rejeté.

F1...15
F2...20
F5...17
F7...5



F1...5
F2...20
F5...17
F7...5

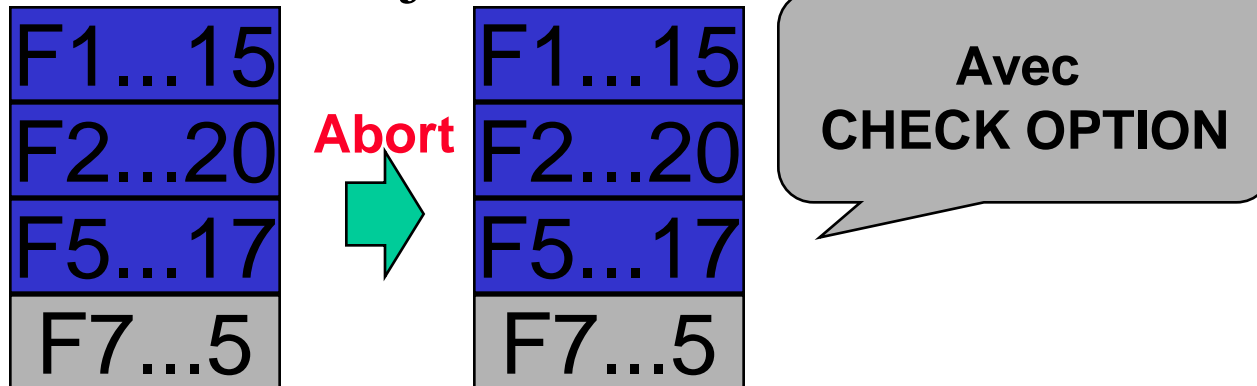
Sans  
CHECK OPTION

## *CHECK OPTION*

- WITH CHECK OPTION protège contre les disparitions de tuples de vues:

UPDATE BONS\_FOURNISSEURS  
SET REMISE = 5 WHERE F = 'F1' ;

serait alors rejeté.





# Mise à jour des vues et ORACLE

- Les conditions qui régissent ces limitations sont les suivantes.
  - Une vue multitable est modifiable (*updatable join view* ou *modifiable join view*) si la requête de définition ne contient pas l'option WITH READ ONLY et la mise à jour (INSERT, UPDATE, DELETE) n'affecte qu'une seule table.
  - Seuls des enregistrements de la table protégée peuvent être insérés (si la clause WITH CHECK OPTION est utilisée, aucune insertion ne sera possible).
  - Seules les colonnes de la table protégée peuvent être modifiées.
  - Seuls les enregistrements de la table protégée peuvent être supprimés.
- Avec Oracle, afin de savoir dans quelle mesure les colonnes d'une table sont modifiables par une vue (insertion, modification ou suppression), il faut interroger le dictionnaire des données par l'intermédiaire de la vue USER\_UPDATABLE\_COLUMNS.

```
SELECT COLUMN_NAME, INSERTABLE, UPDATABLE, DELETABLE  
FROM USER_UPDATABLE_COLUMNS  
WHERE TABLE_NAME = UPPER('bons_fournisseurs');
```

# Mise à jour des vues et SGBD

- Oracle : possibilité de déclarer une vue READ ONLY de manière indépendante à la déclaration des privilèges sur les objets. READ ONLY interdit toute modification.
- SAS admet les mises à jour à travers des vues créée par la PROC SQL. Les privilèges de modification sont définissable au niveau des datasets.
- PostgreSQL : Les vues ont un comportement par défaut READ ONLY. Des règles (RULES) peuvent créer l'effet de vues admettant les mises à jour.
- MySQL jusqu'à la version 5, n'a pas de vues. Elles permettent les mises à jours à travers elles.
- Firebird/Interbase : les mises à jour à travers les vues sont possibles. L'effet de l'option READ ONLY peut être atteint par la non attribution des privilèges autre que SELECT sur les tables de la vue.



# Mise à jour des vues complexes et Oracle

- Un déclencheur `INSTEAD OF` permet de mettre à jour une vue qui ne peut pas être modifiée directement par `INSERT`, `UPDATE` ou `DELETE`. L'expression *instead of* est explicite : le déclencheur programmera des actions au lieu d'insérer, de modifier ou de supprimer dans une vue.
- Ce mécanisme peut être particulièrement intéressant dans un contexte de bases de données réparties où une action sur la base doit être dupliquée via le réseau (*database links*).

**CREATE [OR REPLACE] TRIGGER** name

**INSTEAD OF { UPDATE [OR INSERT] [OR DELETE] }** ON object

**[ REFERENCING NEW AS v1 OLD AS v2 ]**

**FOR EACH ROW**

**DECLARE**

...

**BEGIN**

...

**END;**

L'option de contrôle (`WITH CHECK OPTION`) d'une vue n'est pas vérifiée si un déclencheur `INSTEAD OF` est programmé sur l'événement en question (ajout, modification ou suppression). Le corps du déclencheur doit donc explicitement prendre en compte la contrainte.

name : le nom du déclencheur      object : une vue

Deux nouvelles variables PL/SQL sont disponibles : v1 ou :OLD et v2 ou :NEW

# Exemple de déclencheur INSTEAD OF Oracle

```
CREATE OR REPLACE TRIGGER ins_v_univ INSTEAD OF INSERT ON V_Univ
DECLARE
    compteur INTEGER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('*** INSTEAD OF : INSERT ***');
    IF :OLD.Num_clinique IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('OLD.Num_clinique IS NULL ');
    ELSE
        DBMS_OUTPUT.PUT_LINE('OLD.Num_clinique= ' || TO_CHAR(:OLD.Num_clinique));
    END IF;
    ...
    IF :NEW.Ville IS NOT NULL THEN
        IF :NEW.Departement IS NOT NULL OR :NEW.Nbhab IS NOT NULL THEN
            SELECT COUNT(Ville) INTO compteur FROM T_Implantation WHERE ville=:NEW.Ville;
            IF compteur=0 THEN
                INSERT INTO T_Implantation VALUES(:NEW.Ville,:NEW.Departement,:NEW.Nbhab);
            END IF;
        END IF;
    END IF;
    IF :NEW.Num_Clinique IS NOT NULL THEN
        IF :NEW.Nom IS NOT NULL OR :NEW.Nblits IS NOT NULL OR :NEW.Ville IS NOT NULL THEN
            SELECT COUNT(Num_Clinique) INTO compteur FROM T_Clinique WHERE Num_clinique=:NEW.Num_Clinique;
            ...
        END IF;
    END IF;
END;
/
```

```
-- Permettre l'affichage ...
SET SERVEROUTPUT ON;
BEGIN
    DBMS_OUTPUT.ENABLE(10000);
END;
-- Faire un test
INSERT INTO V_UNIV
VALUES(10,400,'Clairval',70,'Marseille',
'Bouches du Rhône',2000000, 'urgence');
```



# Mise à jour des vues complexes et PostgreSQL

- Il est possible de mettre à jour toutes les vues, soit directement, soit indirectement par l'intermédiaire de règles RULE (qui définissent un comportement en fonction d'évènements et qui sont héritées de Postgres mais toujours de gestion délicate, évitez les boucles sans fin !) avec éventuellement des fonctions FUNCTION pour PostgreSQL

```
CREATE RULE name AS ON event TO object [ WHERE  
condition ] DO [ INSTEAD ] [ action | NOTHING ]
```

name : le nom de la règle                  event : select, update, delete ou insert

object : une table ou une colonne d'une table

condition : toute clause SQL                  action : toute action SQL

Deux nouvelles variables sont disponibles : old et new

```
CREATE RULE ajuster_salaire AS  
    ON UPDATE personnel.salaire WHERE old.nom = "dupont"  
    DO  
  
        UPDATE personnel  
        SET salaire = new.salaire  
        WHERE personnel.nom = "dupond";
```



# *CATALOGUE DU SYSTÈME (1/5)*

## *SPECIFIQUE ORACLE*

Contient sous forme relationnelle la définition de tous les objets créés par le système et les usagers.

Ces tables sont accessibles avec SQL (en mode consultation uniquement).

Les vues commençant par ALL\_ montrent les objets sur lesquels l'utilisateur a des droits.

Les vues commençant par USER\_ donnent les objets sur le compte de l'utilisateur.

Quelques vues utiles gérées par Oracle :

DICTIONARY(DICT) : tous le dictionnaire

USER\_CATALOG (CAT) a pour synonyme CAT et donne les objets contenus sur son compte.

USER\_TABLES (TABS) - tables et vues créées par l'utilisateur

USER\_SYNONYMS (SYN) - Synonymes de l'utilisateur

# *CATALOGUE DU SYSTÈME (2/5)*

## *SPECIFIQUE ORACLE*

USER\_TAB\_COLUMNS (COLS) - colonnes de chaque table ou vue créée par l'utilisateur courant

USER\_INDEXES (IND) - index créés par l'utilisateur courant ou indexant des tables créées par l'utilisateur.

USER\_IND\_COLUMNS – colonnes des index.

USER\_SEQUENCES (SEQ) – séquences

USER\_VIEWS - vues créées par l'utilisateur

USER\_TAB\_GRANTS - objets sur lesquels l'utilisateur est propriétaire, donneur ou receveur d'autorisation

USER\_CONSTRAINTS - définition des contraintes pour les tables de l'utilisateur

USER\_CONS\_COLUMNS - colonnes qui interviennent dans les définitions des contraintes

# *CATALOGUE DU SYSTÈME (3/5) :*

## *SQL-92 INFORMATION\_SCHEMA*

- Base de données introduite dans la norme SQL-92. Composée de plusieurs vues.
- Supportée par MS SQL Server 2000, MySQL 5, PostgreSQL 8 mais pas par Oracle, MS Access, Apache Derby et FireBird. Un [projet](#) open-source implante dans ORACLE ce schema.
- Les principales vues sont :

**TABLES** (table\_schema, table\_name, table\_type) - Toutes les tables et les vues définies dans la base de données.

**COLUMNS** (table\_schema, table\_name, column\_name, ordinal\_position, column\_default, is\_nullable, data\_type) – Informations sur les colonnes.

**VIEWS** (table\_schema, table\_name, view definition) – Toutes les vues.

# *CATALOGUE DU SYSTÈME (4/5) :*

## *AUTRES VUES SQL-92 INFORMATION\_SCHEMA*

<b>Vue</b>	<b>Contenu</b>
SCHEMATA	Nom des schémas du catalogue
TABLE_CONSTRAINTS	Contraintes de table
REFERENTIAL_CONSTRAINTS	Contraintes d'intégrité référentielle
CHECK_CONSTRAINTS	Contraintes de validation : assertions et contrainte de type CHECK de domaine et de table
KEY_COLUMN_USAGE	Colonnes utilisées dans la définition de contraintes PRIMARY KEY, FOREIGN KEY et UNIQUE
TRIGGERS	Déclencheurs
TRIGGER_TABLE_USAGE	Tables utilisées par les déclencheurs
TRIGGER_COLUMN_USAGE	Colonnes utilisées par les déclencheurs
TRIGGERED_UPDATE_COLUMNS	Colonnes utilisées par un déclencheur de mise à jour
VIEW_TABLE_USAGE	Tables utilisées par des vues
VIEW_COLUMN_USAGE	Colonnes utilisées par des vues
CONSTRAINT_TABLE_USAGE	Tables utilisées par une contrainte (référence, unique, assertion)
CONSTRAINT_COLUMN_USAGE	Colonnes utilisées par une contrainte (référence, unique, assertion)

# *CATALOGUE DU SYSTÈME (5/5) :*

## *SPECIFIQUE MySQL*

MySQL suit les recommandations de ANSI/ISO SQL:2003 avec des ajouts et des suppressions. La base de données INFORMATION\_SCHEMA n'admet que l'interrogation.

COLUMNS (...column\_key, column\_comment) – Informations sur les colonnes.

TABLE\_CONSTRAINTS (table\_schema, table\_name, CONSTRAINT\_TYPE, CONSTRAINT\_NAME) - Toutes les contraintes de tables définies dans la base de données.

KEY\_COLUMN\_USAGE (constraint\_schema, constraint\_name, table\_schema, table\_name, column\_name, ordinal\_position, position\_in\_unique\_constraint) – Les colonnes des contraintes.

TRIGGERS (trigger\_schema, trigger\_name, event\_manipulation, event\_object\_catalog, event\_object\_schema, event\_object\_table, action\_order, action\_condition, action\_statement, action\_orientation, action\_timing)

De plus, MySQL continue à enrichir ses commandes spécifiques du dictionnaire de données :

SHOW TABLES, SHOW DATABASES, SHOW VARIABLES

SHOW DATABASES LIKE 'fnuc';

SELECT SCHEMA\_NAME AS `Database` FROM  
INFORMATION\_SCHEMA.SCHEMATA WHERE SCHEMA\_NAME LIKE 'fnuc';

La commande DESCRIBE permet de connaître l'intension d'une table ou d'une vue.

# *VUES ET GROUP BY*

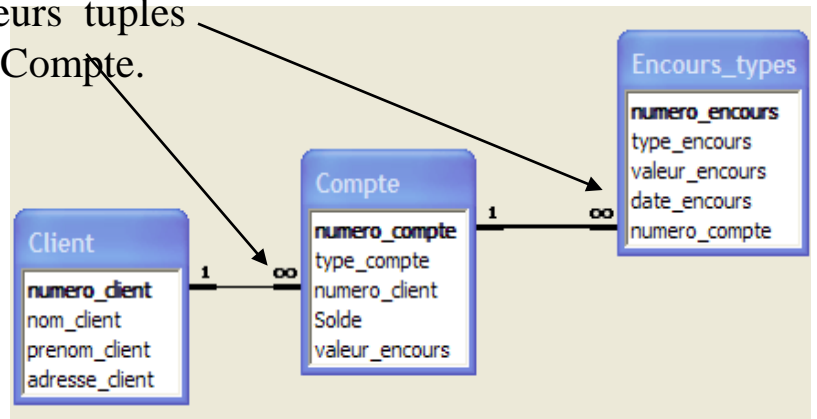
- Les vues pour avoir des attributs calculés
- Les vues pour résoudre le « chasm trap »
- Les vues pour résoudre le « fan trap »

Un tuple Compte  
sera joint avec  
plusieurs tuples de  
Encours\_type.

# VUES ET ATTRIBUTS CALCULES

## BASE DE DONNEES EXEMPLE

De même, un  
tuple de Client  
sera joint avec  
plusieurs tuples  
de Compte.



**Client : Table**

	numero_client	nom_client	prenom_client	adresse_client
+	1	Dubois	Michel	une adresse
+	2	Dupont	Jacques	une autre adresse

Enr : 2 sur 2

**Compte : Table**

	numero_compte	type_compte	numero_client	Solde	valeur_encours
+	1	Compte courant	1	-10,00 €	100,00 €
+	2	Compte courant	2	50,00 €	30,00 €
+	3	Compte epargne	1	100,00 €	-550,00 €
+	4	Compte epargne	1	20,00 €	100,00 €

Enr : 4 sur 4

Solde courant= Solde +

$\sum \text{Encours\_types.valeur\_encours}$

Solde courant= Solde + Compte.  
valeur\_encours

**Encours\_types : Table**

	numero_encours	type_encours	valeur_encours	date_encours	numero_compte
	1	Remise	100,00 €	26/06/2004	1
	2	Retrait	150,00 €	26/06/2004	1
	3	Retrait	100,00 €	27/06/2004	2
	4	Remise	150,00 €	27/06/2004	1
	5	Retrait	150,00 €	27/06/2004	3
	6	Retrait	200,00 €	27/06/2004	1
	7	Remise	50,00 €	28/06/2004	2
	8	Remise	200,00 €	28/06/2004	1
	9	Retrait	250,00 €	28/06/2004	3
	10	Remise	80,00 €	29/06/2004	2
	11	Retrait	100,00 €	29/06/2004	4
	12	Remise	150,00 €	29/06/2004	4
	13	Remise	200,00 €	29/06/2004	3
	14	Retrait	350,00 €	29/06/2004	3
	15	Remise	50,00 €	30/06/2004	4
	16	Remise	100,00 €	30/06/2004	4
	17	Retrait	100,00 €	30/06/2004	4

Enr : 17 sur 17

# CALCULS DES SOLDES DES ENCOURS PAR DES VUES

la restriction des encours aux remises

```
CREATE VIEW Encours_types_Remises AS
SELECT Encours_types.numero_compte, Encours_types.numero_encours, Encours_types.date_encours,
       Encours_types.valeur_encours
FROM Encours_types
WHERE Encours_types.type_encours='Remise'
ORDER BY Encours_types.numero_compte;
```

prenom_client	nom_client	numero_compte	solde_encours
Michel	Dubois	1	100.00 €
Jacques	Dupont	2	30.00 €
Michel	Dubois	3	-550.00 €
Michel	Dubois	4	100.00 €

la restriction des encours aux retraits

```
CREATE VIEW Encours_types_Retraits AS
SELECT Encours_types.numero_compte, Encours_types.numero_encours, Encours_types.date_encours,
       Encours_types.valeur_encours
FROM Encours_types
WHERE Encours_types.type_encours='Retrait';
```

Il faut

**numero\_encours** dans  
les 3 premières vues  
pour éviter  
l'élimination des  
doublons ou alors

les encours signés

```
CREATE VIEW Encours_types_valeurs_signees AS
SELECT Encours_types_Remises.numero_compte, Encours_types_Remises.numero_encours,
       Encours_types_Remises.date_encours, Encours_types_Remises.valeur_encours
FROM Encours_types_Remises
UNION
SELECT Encours_types_Retraits.numero_compte, Encours_types_Retraits.numero_encours,
       Encours_types_Retraits.date_encours, 0- Encours_types_Retraits.valeur_encours as
       valeur_encours
FROM Encours_types_Retraits;
```

**UNION ALL**

les soldes des encours

```
CREATE VIEW Encours_types_Soldes AS
SELECT Client.prenom_client, Client.nom_client, Encours_types_valeurs_signees.numero_compte,
       SUM(Encours_types_valeurs_signees.valeur_encours) AS solde_encours
FROM Client, Compte, Encours_types_valeurs_signees
WHERE Compte.numero_compte=Encours_types_valeurs_signees.numero_compte And
       Client.numero_client=Compte.numero_client
GROUP BY Encours_types_valeurs_signees.numero_compte, Client.prenom_client, Client.nom_client;
```



# *CALCULS DES SOLDES DES ENCOURS PAR DES VUES*

```
CREATE VIEW Encours_types_Remises AS
SELECT Encours_types.numero_compte, Encours_types.date_encours, Encours_types.valeur_encours
FROM Encours_types
WHERE Encours_types.type_encours='Remise'
ORDER BY Encours_types.numero_compte;
```

la restriction des encours aux remises

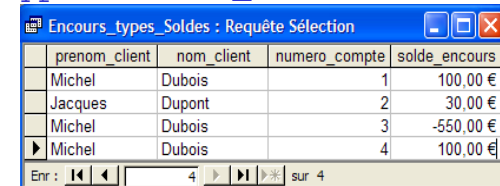
```
CREATE VIEW Encours_types_Retraits AS
SELECT Encours_types.numero_compte, Encours_types.date_encours, Encours_types.valeur_encours
FROM Encours_types
WHERE Encours_types.type_encours='Retrait';
```

la restriction des encours aux retraits

```
CREATE VIEW Encours_types_valeurs_signees AS
```

les encours signés

```
SELECT Encours_types_Remises.numero_compte, Encours_types_Remises.date_encours,
       Encours_types_Remises.valeur_encours
FROM Encours_types_Remises
UNION ALL
SELECT Encours_types_Retraits.numero_compte, Encours_types_Retraits.date_encours,
       0- Encours_types_Retraits.valeur_encours as valeur_encours
FROM Encours_types_Retraits;
```



prenom_client	nom_client	numero_compte	solde_encours
Michel	Dubois	1	100.00 €
Jacques	Dupont	2	30.00 €
Michel	Dubois	3	-550.00 €
Michel	Dubois	4	100.00 €

Plus simple et plus rapide !

```
CREATE VIEW Encours_types_Soldes AS
```

les soldes des encours

```
SELECT Client.prenom_client, Client.nom_client, Encours_types_valeurs_signees.numero_compte,
       SUM(Encours_types_valeurs_signees.valeur_encours) AS solde_encours
FROM Client, Compte, Encours_types_valeurs_signees
WHERE Compte.numero_compte=Encours_types_valeurs_signees.numero_compte And
       Client.numero_client=Compte.numero_client
GROUP BY Encours_types_valeurs_signees.numero_compte, Client.prenom_client, Client.nom_client;
```

# L'EXPRESSION WHEN SQL COMME REMPLACEMENT DE L'UNION DE PLUSIEURS VUES

CREATE VIEW Encours\_types\_valeurs\_signeées AS les encours signés

```
SELECT  Encours_types.numero_compte, Encours_types.numero_encours,  
        Encours_types.Remises.date_encours,  
CASE
```

```
    WHEN Encours_type='Remise'
```

```
        THEN Encours_type.valeur_encours
```

```
    WHEN Encours_type='Retrait'
```

```
        THEN -1*Encours_type.valeur_encours
```

```
    ELSE 0
```

```
END AS valeur_encours
```

```
FROM Encours_types;
```

Elimination de 2 vues, ce qui permet d'éviter de parcourir plusieurs fois la table.

*SAS, PostgreSQL, Oracle 9i supportent l'expression WHEN de la norme SQL.*

CREATE VIEW Encours\_types\_Soldes AS

```
SELECT Client.prenom_client, Client.nom_client,  
        Encours_types_valeurs_signeées.numero_compte,  
        SUM(Encours_types_valeurs_signeées.valeur_encours) AS solde_encours
```

```
FROM Client, Compte, Encours_types_valeurs_signeées
```

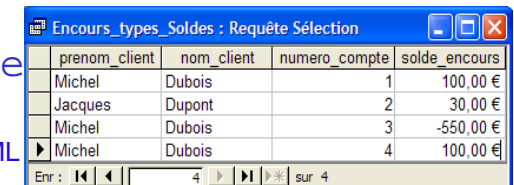
```
WHERE Compte.numero_compte=Encours_types_valeurs_signeées.numero_compte  
And Client.numero_client=Compte.numero_client
```

```
GROUP BY Encours_types_valeurs_signeées.numero_compte,  
        Client.prenom_client, Client.nom_client
```

```
ORDER BY Encours_types_valeurs_signeées.numero_compte
```

## les soldes des encours

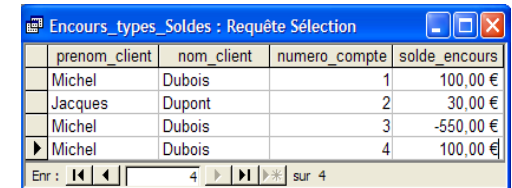
UML



prenom_client	nom_client	numero_compte	solde_encours
Michel	Dubois	1	100,00 €
Jacques	Dupont	2	30,00 €
Michel	Dubois	3	-550,00 €
Michel	Dubois	4	100,00 €

# LA FONCTION DECODE D'ORACLE COMME REMPLACEMENT DE L'UNION DE PLUSIEURS VUES

```
CREATE VIEW Encours_types_valeurs_signees AS les encours signés
SELECT  Encours_types.numero_compte, Encours_types.numero_encours,
        Encours_types_Remises.date_encours,
        DECODE(Encours_type,
                'Remise', Encours_type.valeur_encours,
                'Retrait', -1*Encours_type.valeur_encours,
                0)
        AS valeur_encours
FROM Encours_types;
```



prenom_client	nom_client	numero_compte	solde_encours
Michel	Dubois	1	100,00 €
Jacques	Dupont	2	30,00 €
Michel	Dubois	3	-550,00 €
Michel	Dubois	4	100,00 €

Pour Oracle, la fonction DECODE permet d'obtenir un même résultat. Jusqu'à Oracle 9i, c'est le seul moyen d'avoir un résultat conditionnel dans la clause select.

```
CREATE VIEW Encours_types_Soldes AS les soldes des encours
SELECT Client.prenom_client, Client.nom_client,
        Encours_types_valeurs_signees.numero_compte,
        SUM(Encours_types_valeurs_signees.valeur_encours) AS solde_encours
FROM Client, Compte, Encours_types_valeurs_signees
WHERE Compte.numero_compte=Encours_types_valeurs_signees.numero_compte
And Client.numero_client=Compte.numero_client
GROUP BY Encours_types_valeurs_signees.numero_compte,
        Client.prenom_client, Client.nom_client;
```

# *LA FONCTION MS ACCESS IIF COMME REEMPLACEMENT DE L'UNION DE PLUSIEURS VUES*

La requête sauvegardée **Encours\_types\_valeurs\_signees** : les encours signés

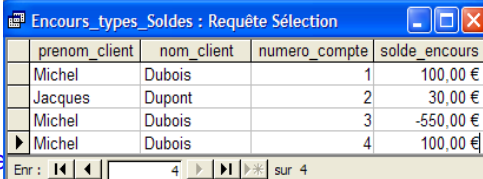
```
SELECT  Encours_types.numero_compte, Encours_types.numero_encours,  
        Encours_types.Remises.date_encours,  
        IIF(Encours_type='Remise', Encours_type.valeur_encours,  
            IIF(Encours_type='Retrait', -1*Encours_type.valeur_encours, 0))  
        AS valeur_encours  
FROM  Encours_types;
```

Elimination de 2 requêtes sauvegardées (ou vue), ce qui permet d'éviter de parcourir plusieurs fois la table.

La fonction MS ACCESS `Iif(<expression>,<resul 1>,<result 2>)` permet d'introduire des conditions dans la clause select. Ceci permet de remplacer une requête avec UNION.

La requête sauvegardée **Encours\_types\_Soldes** : les soldes des encours

```
SELECT  Client.prenom_client, Client.nom_client,  
        Encours_types_valeurs_signees.numero_compte,  
        SUM(Encours_types_valeurs_signees.valeur_encours) AS solde_encours  
FROM  Client, Compte, Encours_types_valeurs_signees  
WHERE  Compte.numero_compte=Encours_types_valeurs_signees.numero_compte  
        And Client.numero_client=Compte.numero_client  
GROUP BY Encours_types_valeurs_signees.numero_compte,  
        Client.prenom_client, Client.nom_client  
ORDER BY Encours_types_valeurs_signees.numero_compte;
```



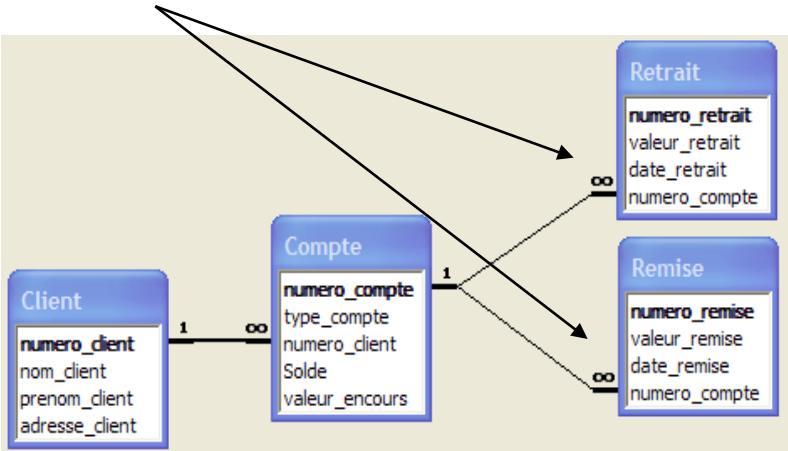
prenom_client	nom_client	numero_compte	solde_encours
Michel	Dubois	1	100.00 €
Jacques	Dupont	2	30.00 €
Michel	Dubois	3	-550.00 €
Michel	Dubois	4	100.00 €

UML pour le

Un tuple Compte sera joint avec plusieurs tuples de Retrait. Il pourra être de plus joint avec plusieurs tuples de Remise.

# GROUP BY ET CHASM TRAP

## BASE DE DONNEES EXEMPLE



Client : Table				
	numero_client	nom_client	prenom_client	adresse_client
+	1	Dubois	Michel	une adresse
+	2	Dupont	Jacques	une autre adresse

Enr : 2 sur 2

Compte : Table					
	numero_compte	type_compte	numero_client	Solde	valeur_encours
+	1	Compte courant	1	-10,00 €	100,00 €
+	2	Compte courant	2	50,00 €	30,00 €
+	3	Compte epargne	1	100,00 €	-550,00 €
+	4	Compte epargne	1	20,00 €	100,00 €

Enr : 4 sur 4

Retrait : Table				
	numero_retrait	valeur_retrait	date_retrait	numero_compte
	1	150,00 €	26/06/2004	1
	2	200,00 €	27/06/2004	1
	3	100,00 €	27/06/2004	2
	4	150,00 €	27/06/2004	3
	5	250,00 €	28/06/2004	3
	6	350,00 €	29/06/2004	3
	7	100,00 €	29/06/2004	4
	8	100,00 €	30/06/2004	4

Enr : 8 sur 8

Remise : Table				
	numero_remise	valeur_remise	date_remise	numero_compte
	1	100,00 €	26/06/2004	1
	2	150,00 €	27/06/2004	1
	3	200,00 €	28/06/2004	1
	4	50,00 €	28/06/2004	2
	5	80,00 €	29/06/2004	2
	6	200,00 €	29/06/2004	3
	7	150,00 €	29/06/2004	4
	8	50,00 €	30/06/2004	4
	9	100,00 €	30/06/2004	4

Enr : 9 sur 9

# GROUP BY ET CHASM TRAP

Quels sont les soldes de chaque compte ?

```
SELECT
```

```
Client.prenom_client, Client.nom_client, Compte.numero_compte,  
    SUM(Remise.valeur_remise)-SUM(Retrait.valeur_retrait) AS  
    solde_encours
```

```
FROM Client, Compte, Remise, Retrait
```

```
WHERE Client.numero_client=Compte.numero_client
```

```
    AND Compte.numero_compte=Remise.numero_compte
```

```
    AND Compte.numero_compte=Retrait.numero_compte
```

```
GROUP BY Compte.numero_compte, Client.prenom_client,  
    Client.nom_client
```

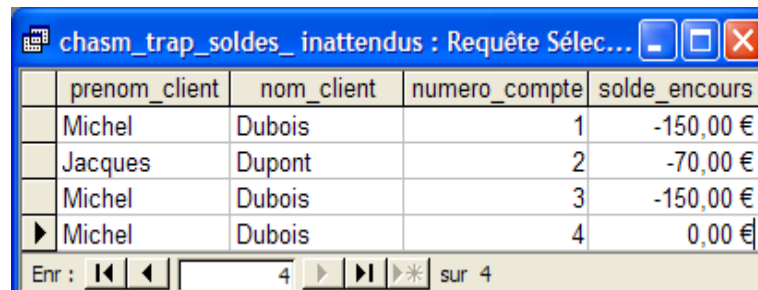
```
ORDER BY Compte.numero_compte;
```

**Résultat**

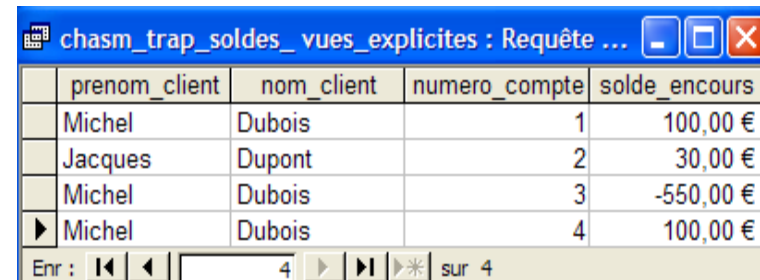
**obtenu**

**Résultat**

**attendu**



prenom_client	nom_client	numero_compte	solde_encours
Michel	Dubois	1	-150,00 €
Jacques	Dupont	2	-70,00 €
Michel	Dubois	3	-150,00 €
Michel	Dubois	4	0,00 €



prenom_client	nom_client	numero_compte	solde_encours
Michel	Dubois	1	100,00 €
Jacques	Dupont	2	30,00 €
Michel	Dubois	3	-550,00 €
Michel	Dubois	4	100,00 €

# GROUP BY, TABLE TEMPORAIRE ET CHASM TRAP

Pour chaque compte, les retraits et les remises ?

```
SELECT Client.prenom_client, Client.nom_client,  
       Compte.numero_compte, Remise.valeur_remise,  
       Retrait.valeur_retrait  
FROM Client, Compte, Remise, Retrait  
WHERE Client.numero_client=Compte.numero_client And  
       Compte.numero_compte=Remise.numero_compte And  
       Compte.numero_compte=Retrait.numero_compte  
ORDER BY Compte.numero_compte
```

## Produit cartésien

prenom_client	nom_client	numero_compte	valeur_remise	valeur_retrait
Michel	Dubois	1	100,00 €	200,00 €
Michel	Dubois	1	150,00 €	150,00 €
Michel	Dubois	1	150,00 €	200,00 €
Michel	Dubois	1	200,00 €	150,00 €
Michel	Dubois	1	200,00 €	200,00 €
Michel	Dubois	1	100,00 €	150,00 €
Jacques	Dupont	2	80,00 €	100,00 €
Jacques	Dupont	2	50,00 €	100,00 €
Michel	Dubois	3	200,00 €	250,00 €
Michel	Dubois	3	200,00 €	350,00 €
Michel	Dubois	3	200,00 €	150,00 €
Michel	Dubois	4	100,00 €	100,00 €
Michel	Dubois	4	150,00 €	100,00 €
Michel	Dubois	4	150,00 €	100,00 €
Michel	Dubois	4	50,00 €	100,00 €
Michel	Dubois	4	50,00 €	100,00 €
Michel	Dubois	4	100,00 €	100,00 €

prenom_client	nom_client	numero_compte	valeur_remise
Michel	Dubois	1	200,00 €
Michel	Dubois	1	150,00 €
Michel	Dubois	1	100,00 €
Jacques	Dupont	2	80,00 €
Jacques	Dupont	2	50,00 €
Michel	Dubois	3	200,00 €
Michel	Dubois	4	100,00 €
Michel	Dubois	4	50,00 €
Michel	Dubois	4	150,00 €

prenom_client	nom_client	numero_compte	valeur_retrait
Michel	Dubois	1	200,00 €
Michel	Dubois	1	150,00 €
Jacques	Dupont	2	100,00 €
Michel	Dubois	3	350,00 €
Michel	Dubois	3	250,00 €
Michel	Dubois	3	150,00 €
Michel	Dubois	4	100,00 €
Michel	Dubois	4	100,00 €

3  
remises

X

2  
retraits



# GROUP BY, VUES EXPLICITES ET CHASM TRAP (1/2)

le total des remises de chaque compte

```
CREATE VIEW Total_remises AS
SELECT DISTINCT Client.prenom_client, Client.nom_client, Compte.numero_compte,
SUM(Remise.valeur_remise) AS total_remise
FROM Client, Compte, Remise
WHERE Client.numero_client=Compte.numero_client And
Compte.numero_compte=Remise.numero_compte
GROUP BY Compte.numero_compte, Client.prenom_client, Client.nom_client
ORDER BY Compte.numero_compte;
```

le total des remises de chaque compte

```
CREATE VIEW Total_retraits AS
SELECT Client.prenom_client, Client.nom_client, Compte.numero_compte,
SUM(Retrait.valeur_retrait) AS total_retrait
FROM Client, Compte, Retrait
WHERE Client.numero_client=Compte.numero_client And
Compte.numero_compte=Retrait.numero_compte
GROUP BY Compte.numero_compte, Client.prenom_client, Client.nom_client
ORDER BY Compte.numero_compte;
```

Total\_remises : Requête Sélection

	prenom_client	nom_client	numero_compte	total_remise
	Michel	Dubois	1	450,00 €
	Jacques	Dupont	2	130,00 €
	Michel	Dubois	3	200,00 €
▶	Michel	Dubois	4	300,00 €

Enr : 4 sur 4

Total\_retraits : Requête Sélection

	prenom_client	nom_client	numero_compte	total_retrait
	Michel	Dubois	1	350,00 €
	Jacques	Dupont	2	100,00 €
	Michel	Dubois	3	750,00 €
▶	Michel	Dubois	4	200,00 €

Enr : 4 sur 4



# GROUP BY, VUES EXPLICITES ET CHASM TRAP (2/2)

SELECT

```
Total_remises.prenom_client, Total_remises.nom_client,  
Total_remises.numero_compte,  
Total_remises.total_remise-Total_retraits.total_retrait  
AS solde_encours
```

FROM

```
Total_remises, Total_retraits
```

WHERE

```
Total_remises.numero_compte=Total_retraits.numero_compte
```

ORDER BY

```
Total_remises.numero_compte;
```

=> Le bon résultat !!!

	prenom_client	nom_client	numero_compte	solde_encours
	Michel	Dubois	1	100,00 €
	Jacques	Dupont	2	30,00 €
	Michel	Dubois	3	-550,00 €
▶	Michel	Dubois	4	100,00 €

Enr : 4 sur 4

1 total retrait

X

1 total remis

=

1 bon solde  
pour le compte

1

	prenom_client	nom_client	numero_compte	total_retrait
	Michel	Dubois	1	350,00 €
	Jacques	Dupont	2	100,00 €
	Michel	Dubois	3	750,00 €
▶	Michel	Dubois	4	200,00 €

Enr : 4 sur 4

	prenom_client	nom_client	numero_compte	total_remise
	Michel	Dubois	1	450,00 €
	Jacques	Dupont	2	130,00 €
	Michel	Dubois	3	200,00 €
▶	Michel	Dubois	4	300,00 €

Enr : 4 sur 4

# GROUP BY, VUES IMPLICITES ET CHASM TRAP

```
SELECT
  Inline_remise.prenom_client, Inline_remise.nom_client,
  Inline_remise.numero_compte, Inline_remise.total_remise-
  Inline_retrait.total_retrait AS solde_encours
FROM (
  SELECT Client.prenom_client, Client.nom_client, Compte.numero_compte,
    SUM(Remise.valeur_remise) AS total_remise
  FROM Client, Compte, Remise
  WHERE Client.numero_client=Compte.numero_client
    AND Compte.numero_compte=Remise.numero_compte
  GROUP BY Compte.numero_compte, Client.prenom_client, Client.nom_client
) AS Inline_remise,
(
  SELECT Retrait.numero_compte, SUM(Retrait.valeur_retrait) AS total_retrait
  FROM Retrait
  GROUP BY Retrait.numero_compte
) AS Inline_retrait
WHERE
  Inline_remise.numero_compte=Inline_retrait.numero_compte
ORDER BY Inline_remise.numero_compte; 1 total retrait
```

=> Le bon solde

	prenom_client	nom_client	numero_compte	solde encours
	Michel	Dubois	1	100,00 €
	Jacques	Dupont	2	30,00 €
	Michel	Dubois	3	-550,00 €
▶	Michel	Dubois	4	100,00 €

X

1 total remis

=

1 bon solde  
pour le compte

1

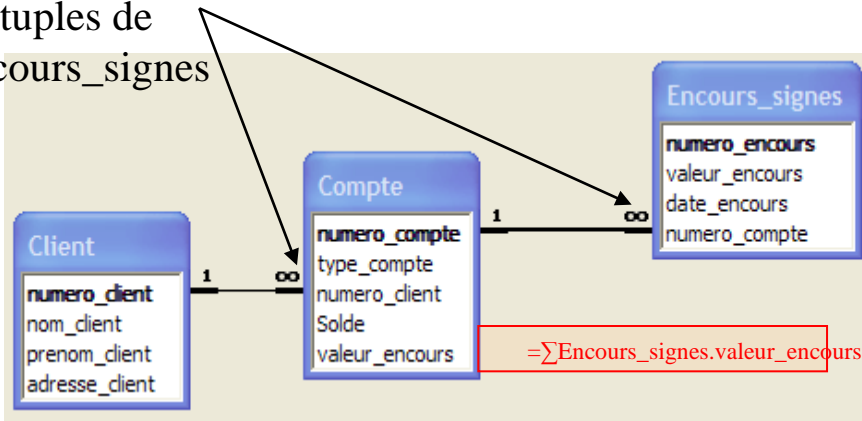
	prenom_client	nom_client	numero_compte	total retrait
	Michel	Dubois	1	350,00 €
	Jacques	Dupont	2	100,00 €
	Michel	Dubois	3	750,00 €
▶	Michel	Dubois	4	200,00 €

	prenom_client	nom_client	numero_compte	total remise
	Michel	Dubois	1	450,00 €
	Jacques	Dupont	2	130,00 €
	Michel	Dubois	3	200,00 €
▶	Michel	Dubois	4	300,00 €

# GROUP BY ET FAN TRAP

## BASE DE DONNEES EXEMPLE

Un tuple de Client  
sera joint avec  
plusieurs tuples de  
Compte. De plus  
un tuple de  
Compte sera joint  
avec plusieurs  
tuples de  
Encours\_signes



	numero_client	nom_client	prenom_client	adresse_client
+	1	Dubois	Michel	une adresse
+	2	Dupont	Jacques	une autre adresse

Enr : 2 sur 2

	numero_compte	type_compte	numero_client	Solde	valeur_encours
+	1	Compte courant	1	-10,00 €	100,00 €
+	2	Compte courant	2	50,00 €	30,00 €
+	3	Compte epargne	1	100,00 €	-550,00 €
+	4	Compte epargne	1	20,00 €	100,00 €

Enr : 4 sur 4

	numero_encours	valeur_encours	date_encours	numero_compte
	1	100,00 €	26/06/2004	1
	2	-150,00 €	26/06/2004	1
	3	-100,00 €	27/06/2004	2
	4	150,00 €	27/06/2004	1
	5	-150,00 €	27/06/2004	3
	6	-200,00 €	27/06/2004	1
	7	50,00 €	28/06/2004	2
	8	200,00 €	28/06/2004	1
	9	-250,00 €	28/06/2004	3
	10	80,00 €	29/06/2004	2
	11	-100,00 €	29/06/2004	4
	12	150,00 €	29/06/2004	4
	13	200,00 €	29/06/2004	3
	14	-350,00 €	29/06/2004	3
	15	50,00 €	30/06/2004	4
	16	100,00 €	30/06/2004	4
	17	-100,00 €	30/06/2004	4

Enr : 17 sur 17

# GROUP BY ET FAN TRAP

Les soldes cumulés d'un client, calcul à partir du résumé et du détail ?

```
SELECT Client.numero_client, Client.prenom_client,
       Client.nom_client, SUM(Compte.valeur_encours) AS solde_compte,
       SUM(Encours_signes.valeur_encours) AS solde_encours
FROM Client, Compte, Encours_signes
WHERE Client.numero_client=Compte.numero_client
      AND Compte.numero_compte=Encours_signes.numero_compte
GROUP BY Client.numero_client, Client.prenom_client,
         Client.nom_client;
```

**Résultat  
obtenu**

=>

Le déficit  
a partir du résumé  
a fortement augmenté

**Résultat  
attendu**

numero_client	prenom_client	nom_client	solde_compte	solde_encours
1	Michel	Dubois	-1 200,00 €	-350,00 €
2	Jacques	Dupont	90,00 €	30,00 €

numero_client	prenom_client	nom_client	solde_compte	solde_encours
1	Michel	Dubois	-350,00 €	-350,00 €
2	Jacques	Dupont	30,00 €	30,00 €

# GROUP BY, TABLE TEMPORAIRE ET FAN TRAP

Pour connaître la table virtuelle, enlever les fonctions d'agrégation et la

Clause GROUP BY :

```
SELECT Client.numero_client, Client.prenom_client,  
       Client.nom_client, Compte.valeur_encours,  
       Encours_signes.valeur_encours
```

```
FROM Client, Compte, Encours_signes
```

```
WHERE Client.numero_client=Compte.numero_client
```

```
AND Compte.numero_compte=Encours_signes.numero_compte;
```

## Table virtuelle résumé et detail

	numero_client	prenom_client	nom_client	Compte.valeur_encours	Encours_signes
	1	Michel	Dubois	100,00 €	100,00 €
	1	Michel	Dubois	100,00 €	-150,00 €
	1	Michel	Dubois	100,00 €	150,00 €
	1	Michel	Dubois	100,00 €	-200,00 €
	1	Michel	Dubois	100,00 €	200,00 €
	2	Jacques	Dupont	30,00 €	-100,00 €
	2	Jacques	Dupont	30,00 €	50,00 €
	2	Jacques	Dupont	30,00 €	80,00 €
	1	Michel	Dubois	-550,00 €	-150,00 €
	1	Michel	Dubois	-550,00 €	-250,00 €
	1	Michel	Dubois	-550,00 €	200,00 €
	1	Michel	Dubois	-550,00 €	-350,00 €
	1	Michel	Dubois	100,00 €	-100,00 €
	1	Michel	Dubois	100,00 €	150,00 €
	1	Michel	Dubois	100,00 €	50,00 €
	1	Michel	Dubois	100,00 €	100,00 €
	1	Michel	Dubois	100,00 €	-100,00 €

L'agrégation de Encours\_signes perturbe celle de valeurs\_encours car leur niveau d'agrégation est différent !

## Table virtuelle résumé seul

	numero_client	prenom_client	nom_client	valeur_encours
	1	Michel	Dubois	100,00 €
	1	Michel	Dubois	-550,00 €
	1	Michel	Dubois	100,00 €
	2	Jacques	Dupont	30,00 €

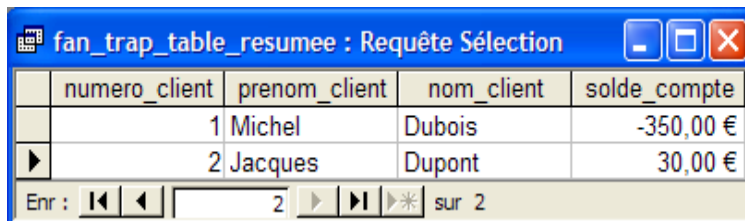
# GROUP BY, VUES EXPLICITES ET FAN TRAP (1/2)

CREATE VIEW fan\_trap\_table\_resumee AS le solde calculé à l'aide de la table  
résumée

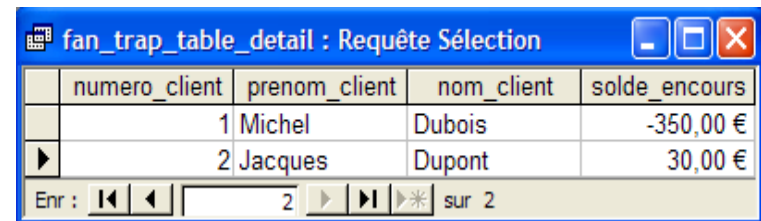
```
SELECT Client.numero_client, Client.prenom_client, Client.nom_client,  
       SUM(Compte.valeur_encours) AS solde_compte  
FROM Client, Compte  
WHERE Client.numero_client=Compte.numero_client  
GROUP BY Client.numero_client, Client.prenom_client, Client.nom_client;
```

CREATE VIEW fan\_trap\_table\_detail AS le solde calculé à l'aide de la table de  
détail

```
SELECT Client.numero_client, Client.prenom_client, Client.nom_client,  
       SUM(Encours_signes.valeur_encours) AS solde_encours  
FROM Client, Compte, Encours_signes  
WHERE Client.numero_client=Compte.numero_client  
      AND Compte.numero_compte=Encours_signes.numero_compte  
GROUP BY Client.numero_client, Client.prenom_client, Client.nom_client;
```



numero_client	prenom_client	nom_client	solde_compte
1	Michel	Dubois	-350,00 €
2	Jacques	Dupont	30,00 €

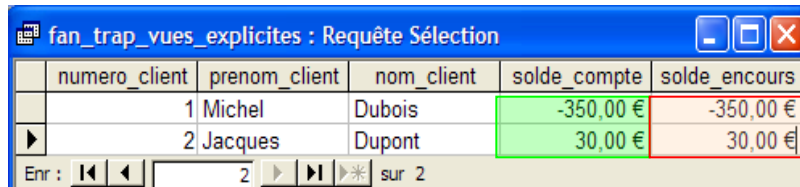


numero_client	prenom_client	nom_client	solde_encours
1	Michel	Dubois	-350,00 €
2	Jacques	Dupont	30,00 €

# GROUP BY, VUES EXPLICITES ET FAN TRAP (2/2)

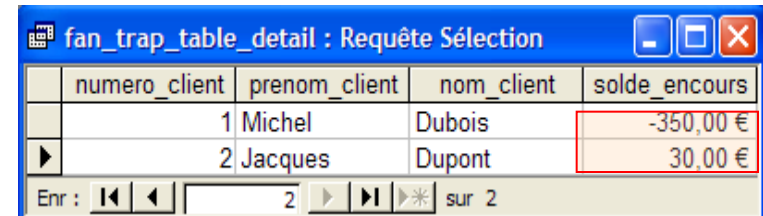
```
SELECT fan_trap_table_resumee.numero_client,  
       fan_trap_table_resumee.prenom_client,  
       fan_trap_table_resumee.nom_client,  
       fan_trap_table_resumee.solde_compte,  
       fan_trap_table_detail.solde_encours  
FROM fan_trap_table_resumee, fan_trap_table_detail  
WHERE fan_trap_table_detail.numero_client=  
  
       fan_trap_table_resumee.numero_client;
```

=> Le bon résultat !!!

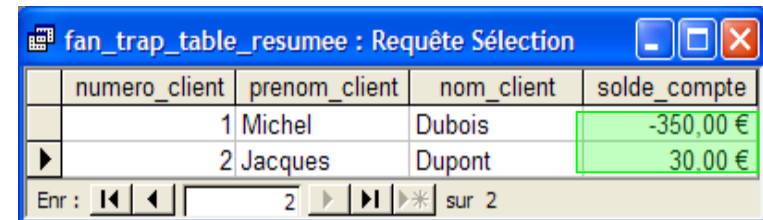


numero_client	prenom_client	nom_client	solde_compte	solde_encours
1	Michel	Dubois	-350,00 €	-350,00 €
2	Jacques	Dupont	30,00 €	30,00 €

Il n'y a plus  
de conflit  
entre  
niveau  
d'agregation



numero_client	prenom_client	nom_client	solde_encours
1	Michel	Dubois	-350,00 €
2	Jacques	Dupont	30,00 €

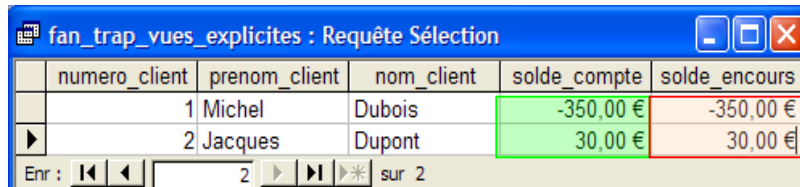


numero_client	prenom_client	nom_client	solde_compte
1	Michel	Dubois	-350,00 €
2	Jacques	Dupont	30,00 €



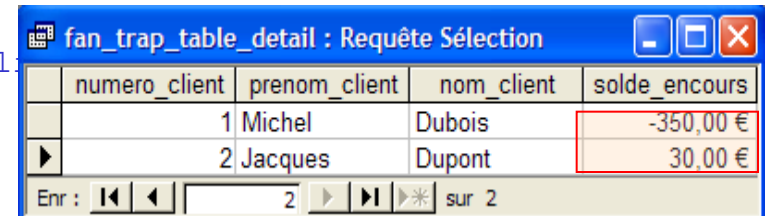
# GROUP BY, VUES IMPLICITES ET FAN TRAP

```
SELECT detail_inline.numero_client, detail_inline.prenom_client, detail_inline.nom_client,  
       detail_inline.solde_encours, resume_inline.solde_compte  
FROM  
(  
    SELECT Client.numero_client, Client.prenom_client, Client.nom_client,  
           SUM(Encours_signes.valeur_encours) AS solde_encours  
    FROM Client, Compte, Encours_signes  
    WHERE Client.numero_client=Compte.numero_client  
           AND Compte.numero_compte=Encours_signes.numero_compte  
    GROUP BY Client.numero_client, Client.prenom_client, Client.nom_client  
) AS detail_inline,  
(  
    SELECT Client.numero_client, Client.prenom_client, Client.nom_client,  
           SUM(Compte.valeur_encours) AS  
           solde_compte  
    FROM Client, Compte  
    WHERE Client.numero_client=Compte.numero_client  
    GROUP BY Client.numero_client, Client.prenom_client, Client.nom_client  
) AS resume_inline  
WHERE detail_inline.numero_client=resume_inline.numero_client  
=> Le bon résultat !!!
```

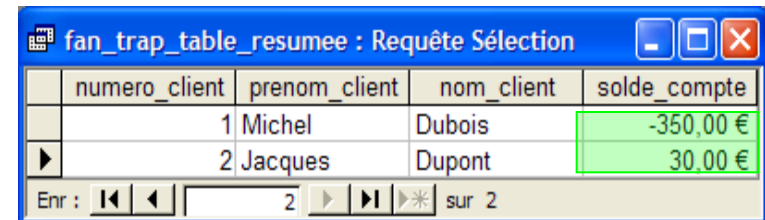


numero_client	prenom_client	nom_client	solde_compte	solde_encours
1	Michel	Dubois	-350,00 €	-350,00 €
2	Jacques	Dupont	30,00 €	30,00 €

Il n'y a plus  
de conflit  
entre  
niveau  
d'agregation



numero_client	prenom_client	nom_client	solde_encours
1	Michel	Dubois	-350,00 €
2	Jacques	Dupont	30,00 €



numero_client	prenom_client	nom_client	solde_compte
1	Michel	Dubois	-350,00 €
2	Jacques	Dupont	30,00 €





# Chapitre 12

## Problèmes liés au contexte Multi-utilisateur

Michel Dubois

I.U.T. de Vannes

*Michel.Dubois@univ-ubs.fr*

[Retour au plan](#)

# *PROBLÈMES LIÉS AU CONTEXTE MULTI-UTILISATEUR*

- Le partage des ressources (inhérent aux bases de données) introduit trois types de problèmes :
- confidentialité et sécurité,
- conflit d'accès concurrents,
- pannes matérielles et logicielles.



# *CONFIDENTIALITÉ ET SÉCURITÉ DANS LES BDR*

Une base de données conduit au partage des informations par plusieurs utilisateurs et programmes. Ce qui implique des risques de

- diffusions ou consultations non réglementaires (problème de confidentialité)
- et de modifications illicites (problème de sécurité)

Ces interventions peuvent être involontaires (accidentelles) ou volontaires (délibérées). Il convient donc d'exercer des contrôles d'accès sur les différents utilisateurs.

# *TYPES DE CONTRÔLE*

On distingue trois types de contrôle sur les accès aux données.

- par le contenant
- par le contexte
- par le contenu

# *CONTRÔLE PAR LE CONTENANT*

Exemple : "Aucun client ne peut accéder à des informations sur les employés".

Solution : SQL permet de définir des droits d'accès sur les objets de la base (tables, vues, etc ...). C'est rapide et efficace.

# *CONTRÔLE PAR LE CONTEXTE*

- Exemple : "Aucun extérieur ne peut accéder à des informations sur les employés sauf si la requête ne permet pas de les identifier".
- Solution : Il faut développer un programme spécifique qui analyse la requête et filtre en fonction du contexte. Ceci demande un travail délicat et ralentit sensiblement l'exécution des requêtes concernées.

# *CONTRÔLE PAR LE CONTENU*

- Exemple : "Un employé ne peut accéder à des informations sur les salaires supérieurs au sien".
- Solution : Il faut développer un programme spécifique qui analyse la requête, consulte un contenu et filtre en fonction de ce contenu. Ceci demande un travail délicat et ralentit fortement l'exécution des requêtes concernées.

# *TYPES DE CONTRÔLE*

On distingue trois types de contrôle sur les accès aux données.

- par le contenant
- par le contexte
- par le contenu

Le coût des contrôles croît selon leur type aussi bien en termes de travail de programmation qu'en termes de ralentissement à l'exécution des requêtes concernées.



# *NIVEAUX DES CONTRÔLES*

- Centralisé
- Réparti
- La tendance actuelle : une solution mixte et syntaxe SQL (dans le cas mixte d'Oracle)

## *NIVEAU CENTRALISÉ*

- On le trouve à l'origine sur les gros systèmes dès les premiers SGBD.
- Il permet un bon contrôle de la sécurité mais nécessite une administration lourde.

## *NIVEAU RÉPARTI*

- Plus récent il apparaît dans la micro et mini informatique.
- La sécurité y est souvent défaillante.
- L'administration de la sécurité est réduite.

# *LA TENDANCE ACTUELLE : UNE SOLUTION MIXTE*

- Une partie des accès est filtrée par l'administrateur qui définit un cadre général (centralisé). une partie est laissée à la responsabilité (limitée) des utilisateurs en fonction de leur compétences estimées par l'administrateur.
- Ce qui allie sécurité (si l'administrateur est vigilant), liberté et responsabilité des utilisateurs.

# *SYNTAXE SQL (DANS LE CAS MIXTE D'ORACLE)*

- Un privilège est le droit attribué à un utilisateur d'exécuter un ensemble donné d'ordres SQL ou d'accéder à certains objets de la base de données.
  - Droits système
  - Droits sur les objets
- Les deux catégories : privilèges système et privilèges sur les objets ne traitent pas des mêmes problèmes.

# *PRIVILÈGE SYSTÈME*

- Il correspond à l'autorisation d'effectuer une action particulière sur la base de données ou sur la définition des objets de la base.
- par exemple, le droit de consulter n'importe quelle table de n'importe quel utilisateur.
- Il y a environ 80 privilèges différents définis pour Oracle 8.
- Ces privilèges sont attribués à chaque utilisateur par l'administrateur de la base de données;

# *EXEMPLES PRIVILÈGES SYSTÈME*

- CREATE TABLE
  - CREATE VIEW
  - CREATE PROCEDURE
  - CREATE TRIGGER
  - CREATE ROLE
  - CREATE SYNONYM
  - CREATE SEQUENCE
  - CREATE ...
- 
- ALTER SESSION
  - ALTER ...

# *SYNTAXE DROITS SYSTÈME*

Accord

**GRANT** {liste de droits système | liste de rôles}  
**TO** {liste d'utilisateurs | PUBLIC} [option]

- Retrait

**REVOKE** {liste de droits système | liste de rôles}  
**FROM** {liste d'utilisateurs | PUBLIC}



# *SCHÉMA DE L 'UTILISATEUR*

- A chaque utilisateur est associe un schéma.
- Un schéma est une liste d'objets contenant :
  - des tables
  - des vues
  - des clusters
  - des procedures
  - des packages
  - ...
- La liste des objets auxquels il accède.

# *PRIVILÈGE OBJET*

- Il correspond à l'autorisation donnée par un créateur d'un objet particulier à un autre ou à un ensemble d'autres utilisateurs.

# *LES DROITS OU PRIVILÈGES OBJET*

- ALTER permet la modification de la définition de l'objet.
- DELETE permet la suppression de tuples.
- EXECUTE autorise l'exécution.
- INDEX autorise la création d'index sur l'objet.
- INSERT autorise l'insertion de tuples.
- REFERENCES autorise la création d'une contrainte de référence sur l'objet.
- SELECT autorise une requête de type SELECT sur l'objet.
- UPDATE autorise la mise à jour des tuples de l'objet.
- ALL tous les droits ci-dessus



# *OBJETS ACCEPTANT CES PRIVILÈGES*

Privilège	Table	Vue	Séquence	Procédure Fonction Package
ALTER	OUI		OUI	
DELETE	OUI	OUI		
EXECUTE				OUI
INDEX	OUI			
INSERT	OUI	OUI		
REFERENCES	OUI			
SELECT	OUI	OUI	OUI	
UPDATE	OUI	OUI		



# *SYNTAXE DROITS SUR LES OBJETS*

- Accord

**GRANT {liste de droits sur objets | ALL}**

**ON objet**

**TO {liste d'utilisateurs | PUBLIC}**

**[option]**

- Retrait

**REVOKE {liste de droits sur objets | ALL}**

**ON objet**

**FROM {liste d'utilisateurs | PUBLIC}**



# *DROITS DE L'UTILISATEUR*

Un utilisateur :

- possède tous les droits sur les objets de son schéma,
- peut donner ou retirer des droits sur les objets de son schéma à des utilisateurs ou à des rôles.

Plusieurs niveaux de privilèges sont possibles selon le type d'objets manipulé.

Pour les opérations DELETE, INSERT, SELECT et UPDATE les privilèges sont attribués au niveau de la table.

Pour INSERT et UPDATE, il est possible de descendre au niveau de la colonnes.



# *RÔLE ORACLE*

- Un rôle est un groupe nommé de privilèges pour des utilisateurs ou des rôles.
- Pour simplifier la gestion des droits :
  - chaque utilisateur possède un rôle,
  - les privilèges sont attribués à un rôle.
- Par défaut tout utilisateur possède le rôle PUBLIC. Ce rôle est utilisé pour donner ou retirer des privilèges à tous les utilisateurs d'ORACLE.

# *RÔLES PRÉDÉFINIS PAR SOUCIS DE COMPATIBILITÉ*

- **CONNECT** : un utilisateur qui a le privilège **CONNECT** peut créer des tables, des vues, des séquences, des clusters, des synonymes et les liens vers d'autres bases de données.
- **RESOURCE** : un utilisateur qui a le privilège **RESOURCE** peut créer ces propres tables, index, séquences, clusters, procédures, fonctions et déclencheurs.
- **DBA** : un utilisateur qui a le privilège **DBA** possède tous les privilèges système nécessaires à l'administrateur de la base de données et la possibilité de les transmettre à d'autres utilisateurs.



# *RÔLE / UTILISATEUR PUBLIC*

Des droits peuvent être accordés à tous les utilisateurs par un seul ordre GRANT en utilisant le mot réservé PUBLIC à la place du nom d'utilisateur

```
GRANT SELECT ON emp TO PUBLIC;
```

- Pour éviter l'utilisation systématique du nom hiérarchique complet pour accéder à un objet, il est possible de créer un synonyme public pour le nom de l'objet par l'ordre CREATE PUBLIC SYNONYM, selon la syntaxe

**CREATE PUBLIC SYNONYM nom synonyme  
FOR [schéma.]objet;**

# Création de l'utilisateur BETTY/BOOPS

**CONNECT SYS/dba247 AS SYSDBA**

**GRANT CONNECT,RESOURCE**

**TO BETTY IDENTIFIED BY BOOPS;**

**ALTER USER BETTY DEFAULT TABLESPACE USERS;**

**ALTER USER BETTY TEMPORARY TABLESPACE TEMP;**

**GRANT CREATE SYNONYM TO betty;**

Création de BETTY  
avec des **privileges**

**CONNECT SCOTT/TIGER**

**GRANT ALL ON EMP TO BETTY;**

**GRANT ALL ON DEPT TO BETTY;**

SCOTT donne tous les **droits**  
sur EMP et DEPT à BETTY

**CONNECT BETTY/BOOPS**

**CREATE SYNONYM EMP FOR SCOTT.EMP;**

**CREATE SYNONYM DEPT FOR SCOTT.DEPT;**

BETTY crée des synonymes pour masquer le nom du propriétaire SCOTT

# *FILTRAGE PAR LES VUES*

Après création d'une vue par une instruction

**CREATE VIEW nom de la vue AS mapping**

- On peut gérer les autorisations d'accès sur les vues plutôt que sur les tables.
- Cependant l'utilisation des vues comporte des limites, on distingue deux cas
  - Vues Monotables : tous les droits en consultation comme en mise à jour sont gérables.
  - Vues MultiTables : seuls les droits en consultation sont gérables.

# *EXEMPLE D'UTILISATION COMBINÉE (1/2)*

```
CREATE ROLE ClassicProf
```

```
GRANT SELECT, INSERT, UPDATE  
ON Notes  
TO ClassicProf
```

```
GRANT ClassicProf  
TO Prof1, Prof2, ...
```

# *EXEMPLE D'UTILISATION COMBINÉE (2/2)*

```
CREATE VIEW NoteEtu AS
SELECT
FROM Notes, Etudiants
WHERE Notes.noEtudiant=Etudiant.NoEtudiant

CREATE ROLE ClassicSec

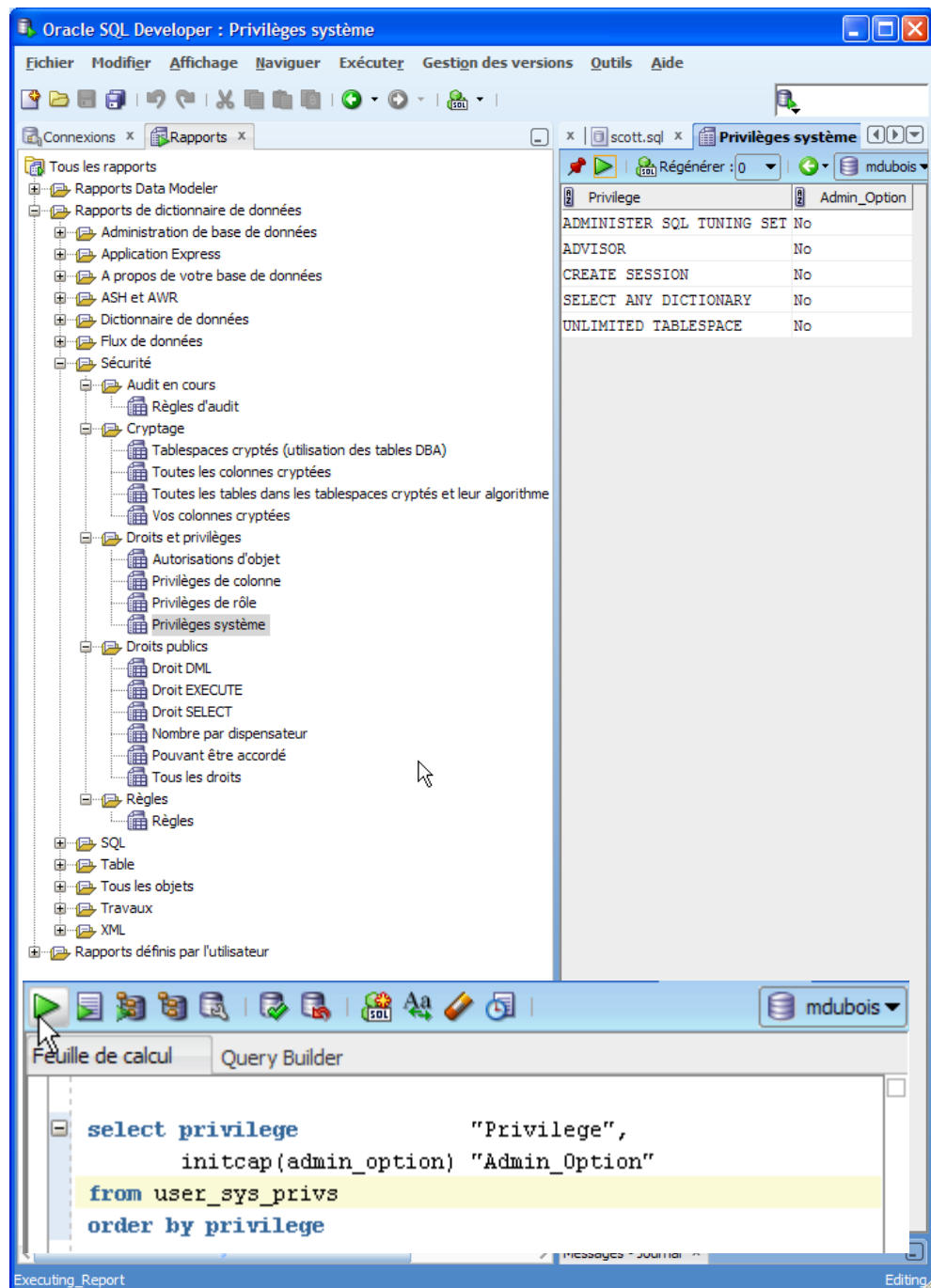
GRANT SELECT
ON NoteEtu
TO ClassicSec

GRANT ClassicSec
TO Sec1, Sec2, ...
```

# RAPPORTS ORACLE SQL DEVELOPER

- La commande Affichage / Rapports met à disposition des rapports directement exécutables pour la connexion courante.
- Parmi ces rapports, au niveau des Rapports de dictionnaire de données, l'entrée Sécurité / Droits et privilèges nous permet de connaître nos privilèges concernant :
  - les privilèges systèmes ;
  - les privilèges de rôles ;
  - les privilèges de colonne.
- Il est toujours possible de voir les requêtes contenues dans un rapport.

Michel Dubois



# *CONCURRENCE ET SGBD*

**Objectif: permettre à plusieurs transactions de travailler en parallèle en donnant à chacune d'entre elle l'illusion de travailler seule.**

Dans cette optique, la notion de transaction correspond à l'unité de cohérence pour une série d'actions exécutées sur la base de données.

Lorsque la série d'action comporte uniquement des opérations de consultation, la notion de transaction permet de garantir que toutes ces actions voient le même état cohérent de la base de données.

Lorsque la série d'actions constituant la transaction comporte des mises à jour, cette notion garantit que la base de données évolue d'un état cohérent vers un nouvel état cohérent, en accord avec la sémantique de la base de données définie notamment par le biais des contraintes d'intégrité.

# *ATOMICITÉ D'UNE TRANSACTION*

Une transaction ne peut s'exécuter qu'en totalité. L'exécution incomplète d'une mise à jour peut laisser la base incohérente si cet état ne viole pas les contraintes d'intégrité.

Une transaction régule les échanges. Elle implique toujours trois acteurs:

- les clients et les fournisseurs sont les acteurs de l'échange. Ils procèdent selon un protocole explicite connu des deux parties.
- le juge arbitre est le garant que la transaction s'est effectuée selon les règles. Soit les règles qui constituent la transaction ont été respectées, et la transaction est déclarée valide, soit elles n'ont pas été respectées et alors la transaction est invalide, c'est à dire qu'il ne s'est rien passé.

Une transaction est un ensemble d'actions cohérentes qui doivent avoir le même sens pour tous les acteurs. C'est donc une notion fondamentalement sémantique.

**Le programmeur doit définir le contenu d'une transaction valide.**

**Le programmeur doit valider ou annuler une transaction.**



# *AUTRES PROPRIÉTÉS ACID*

**Cohérence** : respect des contraintes d'intégrité sur les données. Par contre toute redondance induite au niveau du schéma logique de la base doit être gérée soit par le programmeur, soit par la base. Les indexes sont gérés par le SGBD.

**Isolation** : une transaction ne voit pas les effets des autres transactions concurrentes. Lorsque deux transactions sont exécutées en parallèle, les mises à jour de l'une ne doivent pas être visibles pour l'autre tant que la transaction n'est pas validée.

**Durabilité**: Les effets d'une transaction validée sont permanents. La seule action qui doit permettre de défaire la mise à jour d'une transaction validée est l'exécution d'une transaction de compensation. Ceci implique qu'en cas de panne, des mécanisme de reprise basés sur la notion de transaction valide. En cas de panne, les transactions en cours sont abandonnées.

# *EXEMPLE DE TRANSACTION*

- Dans une agence bancaire si l'on veut faire le transfert d'une somme  $s$  d'un compte de solde  $A$  vers un autre compte de solde  $B$ , on exécutera notamment la suite d'instructions élémentaires suivantes :

lire  $A$ ;

$A \leftarrow A - s$ ;

Ecrire  $A$ ;

lire  $B$ ;

$B \leftarrow B + s$ ;

Ecrire  $B$ ;

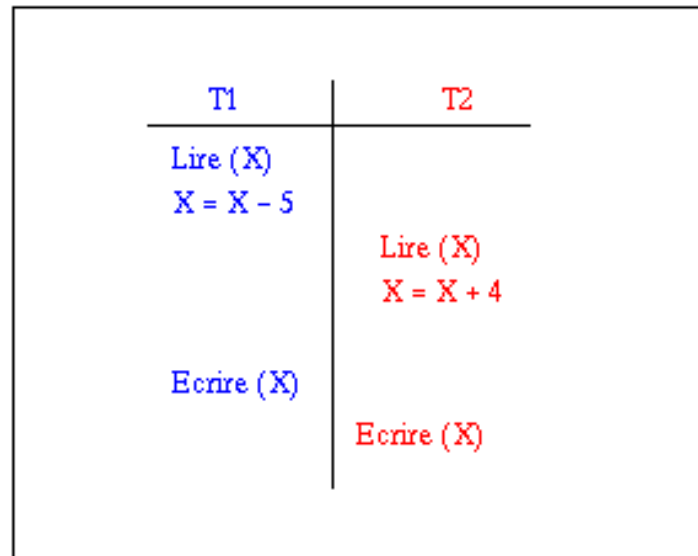
- Or cette suite d'action ne peut être interrompue sans risque d'incohérence (viol de l'invariant  $A+B=\text{constante}$ ).
- Seule la suite d'action est atomique.

# *PROBLÈMES LIÉS À LA CONCURRENCE*

- Perte de mise à jour
- Lecture impropre
- Analyse incohérente
- Objets fantômes : des objets sont apparus !

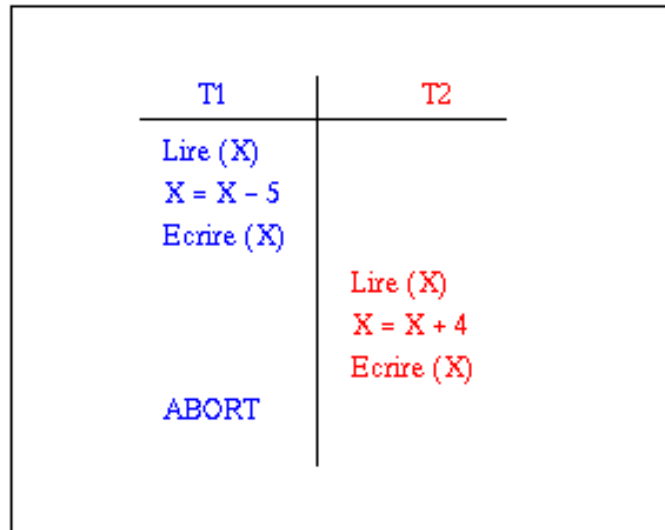
# *PERTE DE MISE À JOUR*

- La perte de la mise à jour d'une donnée peut avoir lieu lorsque deux transactions qui accèdent aux mêmes données ont leurs opérations imbriquées, si bien qu'une valeur d'une donnée devient incorrecte.
- L'anomalie de perte de mise à jour résulte de la dépendance Ecriture-Ecriture entre actions de plusieurs transactions interférentes.



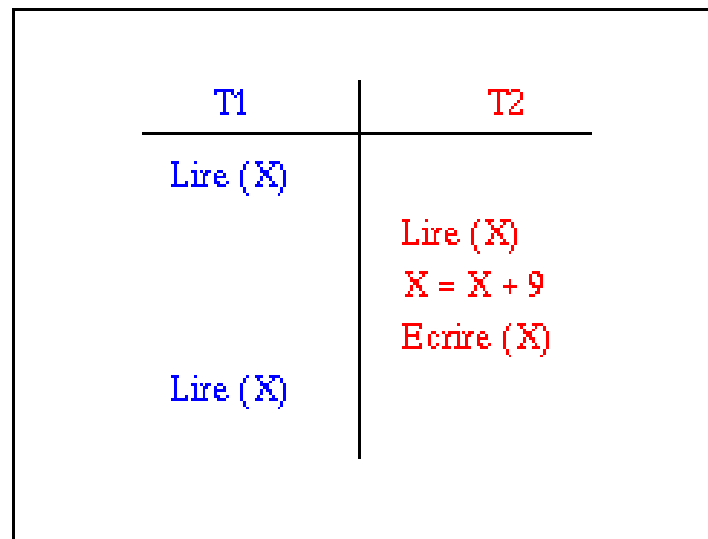
# *LECTURE IMPROPRE (DIRTY READ)*

- Une lecture d'une valeur temporaire de donnée peut conduire à une valeur fausse.
- Ceci a lieu lorsqu'une transaction met à jour une donnée de la base et que cette transaction échoue ou est annulée. La valeur mise à jour est lue par une seconde transaction avant que la donnée ait retrouvé sa valeur initiale conduisant à un "dirty read".
- L'anomalie de lecture impropre résulte de la dépendance Ecriture-Lecture entre actions de différentes transactions.



# *LECTURE NON REPRODUCTIBLE*

- Dans le corps de la même transaction, la relecture d'une donnée peut produire un résultat différent du fait de la modification de la donnée par une autre transaction entre les deux lectures successives.
- L'anomalie de lecture non reproductible est un cas particulier de lecture incohérente due à l'observation d'états momentanément incohérents. Cette anomalie qui résulte de la dépendance Lecture-Ecriture entre actions correspond à une destruction de l'intégrité interne des données.



# *OBJETS FANTÔMES*

- Ce problème survient quand une transaction n'a pas perçu la création d'un tuple par une autre transaction.
- Par exemple, une transaction lit d'abord le nombre d'employés et lance ensuite la lecture d'informations sur chacun de ces employés
- Si une autre a ajouté des employés, le nombre lu auparavant ne correspond plus à la réalité.

# *QUELQUES RÉFLEXIONS SUR LA RÉSOLUTION DES CONFLITS*

- Identification des conflits
- Stratégies extrêmes
- Objectif : sérialiser
- Mécanismes de résolution



# *IDENTIFICATION DES CONFLITS*

- Il y a un conflit entre deux transactions T1 et T2 si et seulement si les deux conditions suivantes sont réalisées :
  - T 1 et T2 accèdent à la même source de données,
  - T 1 et T2 se chevauchent dans le temps.
- La « même source » peut être une table ou un tuple. On parlera de granule, c 'est à dire l'unité d'accès indivisible (zone dont il faut contrôler l'accès afin d 'éviter les conflits).

# *OBJECTIF DE LA GESTION DES CONFLITS*

- Stratégies extrêmes
  - exécution en série : pas de conflit, mais trop lent
  - exécution en parallèle : trop de conflit
- Objectif : sérialiser

C'est-à-dire exécuter le plus possible en parallèle en garantissant les mêmes résultats que lors d'une exécution séquentielle.

# EXÉCUTION EN PARALLÈLE VERSUS EN SÉRIE

T1	T2	T1	T2
Lire(X) X:=X-10 Ecrire(X) Lire(Y) Y=Y+10 Ecrire(Y)	      Lire(X) X=X+20 Ecrire(X)	      Lire(X) X:=X-10 Ecrire(X) Lire(Y) Y=Y+10 Ecrire(Y)	      Lire(X) X=X+20 Ecrire(X)

Schema 1: Transactions en serie

T1	T2	T1	T2
Lire(X) X:=X-10   Ecrire(X) Lire(Y)   Y=Y+10 Ecrire(Y)	      Lire(X) X=X+20   Ecrire(X)	      Lire(X) X:=X-10 Ecrire(X)   Lire(Y) Y=Y+10 Ecrire(Y)	      Lire(X) X=X+20 Ecrire(X)

Schema 2: Execution en parallele



# PRINCIPE DU VERROU

- Pour prévenir les risques de conflit une transaction peut verrouiller les données qu'elle manipule ou plus exactement les granules contenant ces données. Par verrouillage cette transaction peut interdire ou limiter l'accès aux granules concernés.
- Il existe deux types de verrou : les verrous de lecture (*partagés*) et les verrous d'écriture (*exclusifs*).
  - Lorsqu'une transaction pose un verrou de lecture, sur le granule g, toutes les autres transactions peuvent aussi poser un verrou de lecture sur g, par contre aucune autre transaction ne peut poser de verrou en écriture sur g.
  - Lorsqu'une transaction pose un verrou d'écriture, sur le granule g, aucune autre transaction ne peut poser de verrou sur g (ni en lecture, ni en écriture).
- Les transactions qui se voient interdire l'accès à un granule sont suspendues et placées dans la file d'attente de ce granule.
- Pour simplifier dans la suite on ne distinguera pas ces deux types de verrous, tout verrou se comportera comme un verrou en écriture.

# VERROUILLAGE D'UN GRANULE

- Soit  $T_i$  la transaction demandant le verrouillage du granule  $g$ .  
Le système exécute alors un algorithme de verrouillage comparable au suivant :

Algorithme Verrouillage ( $g, i$ )

début

    si  $g$  n'est pas déjà verrouillé alors

        mettre un verrou sur  $g$  pour  $T_i$  ;

    sinon

        mettre  $i$  dans la file d'attente du granule  $g$  ;

        suspendre l'exécution de  $T_i$ ;

    fin-si

fin

	Verrou X présent	Verrou(s) S présent(s)	Pas de verrou présent
Verrou X demandé	Demande refusée	Demande refusée	Demande accordée
Verrou S demandé	Demande refusée	Demande accordée	Demande accordée

OK pour les cas de demandes p. 629

# *DEVERROUILLAGE D'UN GRANULE*

- Inversement quand **Ti** n'utilise plus les ressources du granule **g**, le système le libère afin que les demandes en attente puissent être satisfaites. Pour cela il exécute un algorithme de déverrouillage comparable au suivant :

Algorithme Déverrouillage (g,i)

début

supprimer le verrou de Ti sur g

pour tout indice j de la file d'attente de g répéter

si g n'est pas verrouillé alors

enlever j de la file d'attente

débloquer l 'exécution de Tj

fin-si

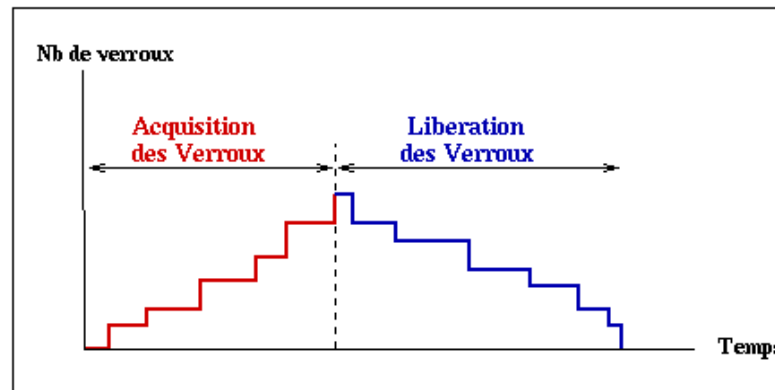
fin-pour

fin

- En fait ces algorithmes ne sont que des ébauches, ils ne supporteraient pas l'abandon de l'hypothèse simplificatrice de fusion des verrous de lecture et d'écriture.

# VERROUILLAGE A DEUX PHASES

- Les accès conflictuels sont détectés au moyens de verrous posés par les transactions sur les granules contenant les objets accédés.
- Toute transaction comporte deux phases distinctes :
  - L'acquisition des verrous lors des accès conflictuels.
  - Libération des verrous à la fin de la transaction.



- Les transaction exécutées selon ce principe sont sérialisables
- Le problème majeur lié à l'utilisation du verrouillage à deux phases est le problème d'inter-blocage.

# *EXEMPLE D'UN VERROUILLAGE À 2 PHASES*

Soit la séquence d'opérations ci dessous.

H=  $r_1[x]$   $r_2[y]$   $w_3[x]$   $w_1[y]$   $w_1[x]$   $w_2[y]$   $c_2$   $r_3[x]$   $r_1[y]$   $c_1$   $w_3[y]$   $c_3$

Le gestionnaire de transaction va bloquer certaines opérations en attente d'un verrou. Dès que le verrou devient disponible, l'opération s'exécute. Les verrous d'une transaction sont relâchés au moment du commit.

$r_1[x]$   $r_2[y]$  exécutées

$w_3[x]$  bloquée à cause de  $r_1[x]$

$w_1[y]$  bloquée à cause de  $r_2[y]$

$w_1[x]$  bloquée à cause de  $w_1[y]$

$w_2[y]$   $c_2$   $w_1[y]$   $w_1[x]$

$r_3[x]$  bloquée à cause de  $w_3[x]$  (T1 n'a pas encore libéré ses verrous !)

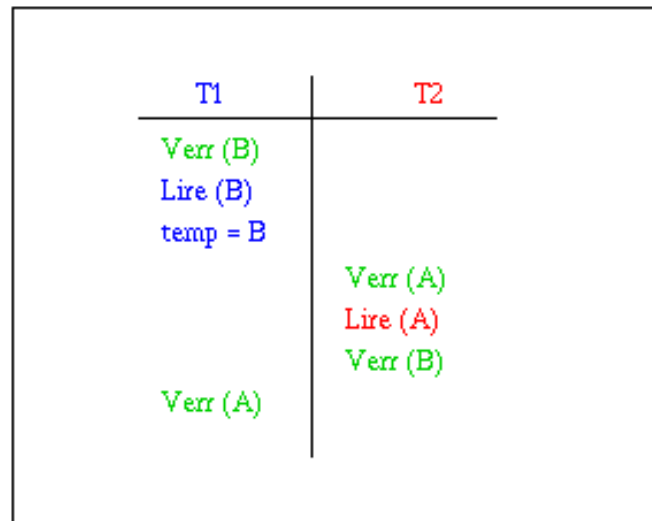
$r_1[y]$   $c_1$   $w_3[x]$   $r_3[x]$   $w_3[y]$   $c_3$

Michel Dubois



# NOTION D 'INTERBLOCAGE

- Comme T1 verrouille B et que T2 demande un verrouillage de B, T2 attend que T1 déverrouille B. De même, comme T2 verrouille A et T1 demande un verrouillage de A, T1 attend que T2 déverrouille A.
- Nous avons une situation où aucune de ces deux transactions ne peut progresser. Il y a un verrou mortel.



# DEADLOCKS OU INTERBLOCAGE

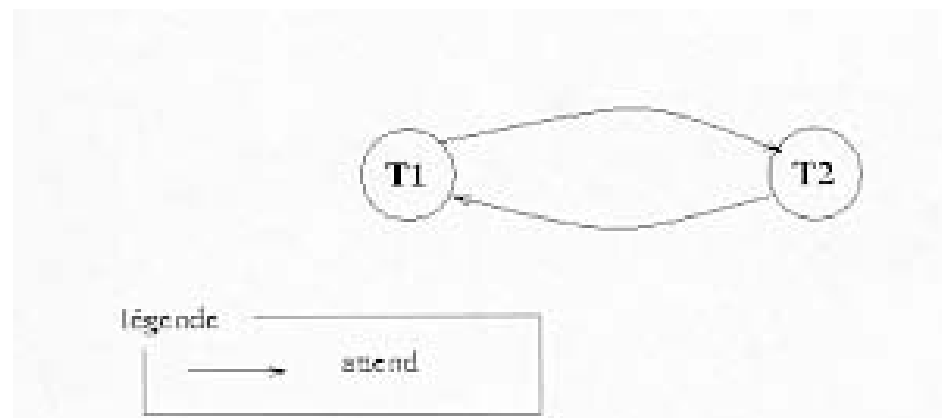
Transaction 1	Time	Transaction 2
<code>UPDATE emp SET sal = sal * 1.1 WHERE empno = 1000;</code>	1	<code>UPDATE emp SET mgr = 1342 WHERE empno = 2000;</code>
<code>UPDATE emp SET sal = sal * 1.1 WHERE empno = 2000;</code>	2	<code>UPDATE emp SET mgr = 1342 WHERE empno = 1000;</code>
<code>ORA-0060: deadlock detected while waiting for resource</code>		
<code>ROLLBACK;</code>		

La seconde transaction 3 détecte le verrou mortel. Oracle annule l'instruction qui l'a détectée et lui envoie un message. La transaction n'est pas annulée pour autant.

Oracle identifie et résout les verrous de ce type en annulant l'instruction qui les a détectés.

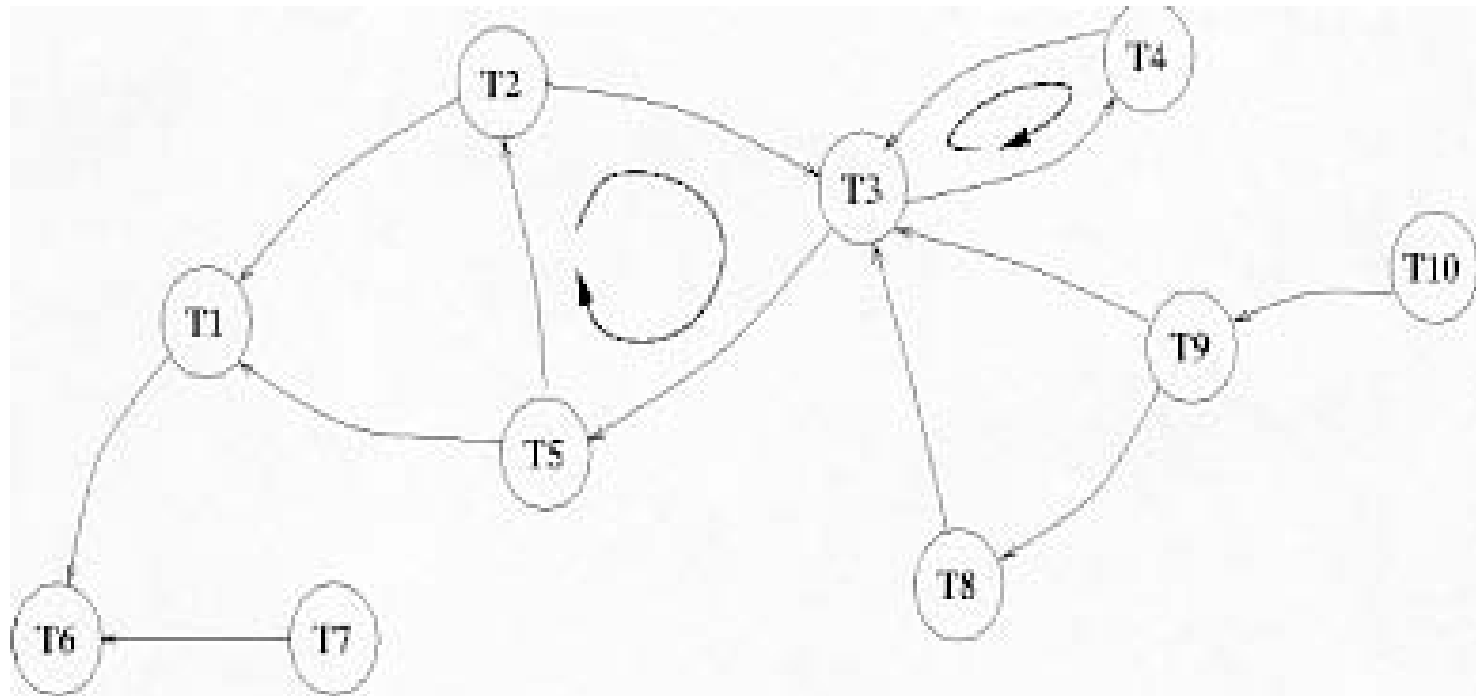
# La détection des interblocages

- Elle s'effectue au niveau du graphe des attentes. Par exemple l'interblocage de deux transactions correspond à :



# *EXEMPLE DE GRAPHE DES ATTENTES*

- Dans ce graphe T6, T7 et T1 seront exécutées sans problème. Par contre T2, T5 et T3 sont en interblocage et bloquent de ce fait toutes les transactions qui les attendent.



# Situation d'étreinte fatale

```
1 SCOTT> UPDATE emp  
      SET sal=sal*1.1  
      WHERE empno=7839;
```

1 row updated.

```
3 SCOTT> UPDATE emp  
      SET sal=sal*1.1  
      WHERE empno=7369;
```

```
ERROR at line 1:  
ORA-00060: deadlock detected  
while waiting for resource
```

```
5> ROLLBACK;
```

SCOTT

```
2 BETTY> UPDATE emp  
      SET job='ANALYST'  
      WHERE empno=7369;
```

1 row updated.

```
4 BETTY> UPDATE emp  
      SET job='CEO'  
      WHERE empno=7839;
```

La seconde **transaction 3>** détecte le verrou mortel. Oracle annule l'instruction qui l'a détectée et lui envoie un message. La transaction n'est pas annulée pour autant.

1 row updated.

BETTY

# Exercice: comment éviter les verrous mortels?

1. Ouvrir deux sessions sous **SCOTT**
2. Ecrire une procédure PL appelée **batch** qui augmente de **10%** le salaire d'un employé. Celui-ci est spécifié par son matricule **empno** passé en paramètre (**p\_empno**) .

Cette procédure lève l'exception **ORA-00060 (PRAGMA)**.

Si une étreinte fatale est détectée, un **ROLLBACK** aura lieu.

3. Commencez par noter le salaire des employés **7839** et **7369**
4. Effectuez les gestes suivants:

a) **Session 1: SQL> EXEC batch(7839);**

b) **Session 2: SQL> EXEC batch(7369);**

c) **Session 1: SQL> EXEC batch(7369);**

d) **Session 2: SQL> EXEC batch(7839);**

- L'étreinte, a-t-elle été résolue?
- Quelle session a résolu le verrou mortel?

# Détecter les verrous mortels ..

```
CONNECT scott/tiger
```

```
CREATE OR REPLACE PROCEDURE batch (p_empno NUMBER)
IS
    DEAD_LOCK    EXCEPTION;
    PRAGMA        EXCEPTION_INIT(DEAD_LOCK,-00060);
BEGIN
    DBMS_OUTPUT.ENABLE(32000);
    UPDATE emp SET sal = sal * 1.1 WHERE empno = p_empno;
    DBMS_OUTPUT.PUT_LINE(p_empno || 'mis à jour');
EXCEPTION
    WHEN DEAD_LOCK THEN
        DBMS_OUTPUT.PUT_LINE('Etreinte résolue: ' || p_empno);
        ROLLBACK;
END;
/
```

```
GRANT EXECUTE ON batch TO BETTY;
CREATE PUBLIC SYNONYM batch FOR scott.batch;
```

Notez les salaire des matricules **7839** et **7369**

```
SQL> CONNECT scott/tiger
```

```
SQL> SELECT empno,sal  
        FROM emp  
        WHERE empno in (7839,7369);
```

EMPNO	SAL
7369	800
7839	5000

Notez ces valeurs



# Provoquez l'étreinte fatale

<1> EXEC batch (7839);  
7839 mis à jour.

<3> EXEC batch (7369);  
➤ Bloqué..

ORA-00060 en cours de détection

7369 mis à jour  
PL/SQL procedure successfully completed.

SCOTT

Pas de problème pour le moment

<2> EXEC batch (7369);  
7369 mis à jour.

<4> EXEC batch (7839);  
➤ Bloqué

7839 mis à jour  
PL/SQL procedure successfully completed.

BETTY

Temps

# Le verrou mortel a-t-il été résolu?

```
SQL> CONNECT scott/tiger
```

```
SQL> SELECT empno,sal
```

```
      FROM emp
```

```
      WHERE empno in (7839,7369);
```

EMPNO	SAL
7369	800
7839	5000

On retrouve les mêmes valeurs.  
ORA-00060 détectée et résolue



# DÉROULEMENT D'UNE TRANSACTION ORACLE

Si l'**AUTOCOMMIT** est activé (**ON**): chaque instruction constitue une transaction.

On multiplie les échanges entre mémoire principale et mémoire secondaire.

Sinon (**AUTOCOMMIT OFF**) :

Une transaction débute

lors de l'exécution d'une instruction de définition de données (CREATE, ALTER, DROP, RENAME) et de manipulation de données (SELECT, INSERT, UPDATE, DELETE).

Au cours de la transaction (cas du traitement par lot) :

Des point de reprise peuvent définis par **SAVEPOINT pr0;**

Une reprise peut alors s'effectuer par **ROLLBACK TO SAVEPOINT pr0;**

Une transaction se termine par :

- un ordre explicite (**COMMIT;** ou **ROLLBACK;** ) pour les applications.
- un ordre de définition de données LDD valide la transaction et en démarre une nouvelle qui est validée en cas de succès.
- une déconnexion normale d'un utilisateur valide la transaction. Dans le cas contraire, elle est invalidée.

# *ORACLE ET PARALLÉLISME AUTORISÉ*

Le degré de parallélisme autorisé par un SGBD dépend de trois facteurs :

- le granule de contrôle de concurrence;
- la stratégie de contrôle de concurrence;
- le degré d'isolation des transactions.

# *ORACLE ET PARALLÉLISME AUTORISÉ*

- La stratégie de contrôle de concurrence d'Oracle est un mécanisme de verrouillage à deux phases.
- Les accès en écritures par deux transactions sur un même objet sont conflictuels.
- Par défaut, Oracle ne met pas de verrou pour les SELECT.

Le standard **ANSI / ISO SQL92** a défini trois sortes d'interférences possibles entre les transactions, et quatre niveaux d'isolation pour élever la cohérence contre de telles interactions:

Niveau d'isolation	Dirty Read	Non-Repeatable Read	Phantom Read
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	Pas possible	Possible	Possible
REPEATABLE READ	Pas possible	Pas possible	Possible
SERIALIZABLE	Pas possible	Pas possible	

Par défaut  
READ COMMITTED

# *ORACLE ET PARALLÉLISME AUTORISÉ*

Dans le but d'augmenter le nombre de transactions pouvant être exécutées en parallèle, Oracle propose deux niveaux d'isolation.

- Le mode par défaut, `READ COMMITED` garantit qu'une transaction ne voit que des données qui ont été validées ou qui sont en cours de mise à jour par elle-même. En revanche, il n'interdit pas à d'autres transactions de modifier les données qui viennent d'être consultées par une transaction en cours. Ainsi, les données consultées par une telle transaction sont cohérentes sur une requête mais pas sur l'ensemble de la transaction. (ISO `READ_COMMITED`)
- Le mode `SERIALIZABLE` garantit qu'une même requête exécutée plusieurs fois par la même transaction retrouvera toujours le même résultat, excepté si cette transaction a elle même modifiée les données du résultat. (ISO `REPEATABLE READ`)

# L'importance du niveau d'isolation

- **SCOTT** a pris en compte la transaction validée (**COMMIT**) de **BETTY**.
- **SCOTT** voit ainsi que la somme des salaires diffère. C'est du reste logique (consistance).
  - Comment, dans ces conditions, effectuer un arrêté comptable de fin de mois (il finit à minuit légalement) si le batch s'arrête avant minuit ou si le batch finit après minuit fin de mois?
- Réponse: changer le niveau d'isolation au début du batch de consolidation comptable.



# Consolidation comptable

<S> SET TRANSACTION t0  
READ ONLY; Minuit

<S> SELECT SUM(sal) t1  
FROM emp  
GROUP BY deptno;

SUM(SAL)

8750

10875

9400

<S> SELECT SUM(sal) t4  
FROM emp;

SUM(SAL)

29025

SCOTT

Au COMMIT, Scott  
obtiendra 30025

Cas réel

<B> UPDATE emp  
SET sal=sal\*1.2  
WHERE empno=7839;

1 row updated.

<B> commit;

<B> SELECT SUM(sal)  
FROM emp;

SUM(SAL)

30025

BETTY

↓ Au matin

# Multi versions et multi utilisateurs

- En concurrence d'accès à la base:
  - Une lecture ne doit pas empêcher une écriture
  - Une lecture ne doit pas empêcher une lecture
  - Une écriture ne doit pas empêcher une lecture
  - Une écriture ne doit pas empêcher une écriture
    - Une écriture bloque une écriture qui porte sur la sur la même ligne
- Dans ce cas, la base est réputée OLTP
  - On Line Transactional Processing (Haut débit transactionnel)

# *SOLUTION DES PROBLÈMES AVEC ORACLE*

- Perte de mise à jour
- Lecture impropre
- Analyse incohérente
- Objets fantômes

# *PROBLÈMES DE PERTE DE MISE À JOUR*

Soit un compte x ayant en crédit 200 €:  $x=200$

$T_1 = \text{crédit } x \text{ de } 100 \quad F = r_1[x]w_1[x]c_1$

$T_2 = \text{crédit } x \text{ de } 50 \quad F = r_2[x]w_2[x]c_2$

$H = r_1[x]_{x:200} \quad r_2[x]_{x:200} \quad w_1[x]_{x:300} \quad w_2[x]_{x:250} \quad c_1 \quad c_2$

Résultat :  $x=250$  au lieu de  $x=350$  ( $w_1[x]$  est perdu à cause de  $w_2[x]$ )

Une solution à ce problème est de verrouiller l'accès en lecture en vue d'une mise à jour par l'ordre :

SELECT ... FOR UPDATE ou par LOCK TABLE ...

# *LECTURE IMPROPRE*

Soir un compte x au solde créditeur de 200 €:  $x=200$

$T_1 = \text{crédit } x \text{ de } 100 \text{ €} \leftarrow r_1[x]w_1[x]c_1$

$T_2 = \text{crédit } x \text{ de } 50 \text{ €} \leftarrow r_2[x]w_2[x]c_2$

$H = r_1[x]_{x:200} w_1[x]_{x:300} r_2[x]_{x:300} w_2[x]_{x:350} c_2 a_1$

Résultat :  $x=350$  au lieu de  $x=250$  ( $T_1$  est annulée)

Oracle garantit qu'une transaction ne peut voir les effets d'autres transactions que si elles ont été validées. Par contre elle n'est pas prévenue dans le niveau ISO TRANSACTION NONE & TRANSACTION READ UNCOMMITTED.

# *ANALYSE INCOHÉRENTE*

Soient deux comptes appartenant à Personne

le solde créditeur de x est de 200 €:  $x=200$

le solde créditeur de y est de 100 €:  $y=100$

$T_1$ =transfert de 50 F de x vers y= $r_1[x]w_1[x] r_1[y]w_1[y] c_1$

$T_2$ =calcul dans z de la somme  $x+y=r_2[x] r_2[y] w_2[z] c_2$

$H= r_1[x]_{x:200} w_1[x]_{x:150} r_2[x]_{x:150} r_2[y]_{y:100} w_2[z]_{z:250} c_2 r_1[y]_{y:100} w_1[y]_{y:150} c_1$

Résultat :  $z=250$  au lieu de  $z=300$  ( $r_2[x]$  est influencée par  $T_1$  mais  $r_2[y]$  non)

Oracle garantit qu'une transaction ne peut voir les effets d'autres transactions que si elles ont été validées.

# *LECTURE NON RÉPÉTITIVE*

Soir un compte  $x$  au solde créditeur de 200 €:  $x=200$

$T_1$ =plusieurs consultations du solde de  $x=r_1[x] \dots r_1[x]$

$T_2$ =crédit  $x$  de 50 €  $=r_2[x]w_2[x]c_2$

$H = r_1[x]_{x:200} r_2[x]_{x:200} w_2[x]_{x:250} c_2 r_1[x]_{x:250}$

Résultat : la valeur de  $x=350$  a changé

Pour éviter cela, la transaction 1 devrait mettre des verrous partagés sur les tuples consultés. Avec l'instruction LOCK TABLE . Oracle propose aussi de déclarer  $T_1$  comme une transaction READ ONLY.

# OBJETS FANTÔMES

Soient deux comptes appartenant à Personne

le solde créditeur de x est de 200 €:  $x=200$

le solde créditeur de y est de 100 €:  $y=100$

la somme des comptes de M. Personne est contenue dans s:  $s=300$

$T_1$ =calcul somme comptes Personne et comparaison avec s =  $r_1[x]r_1[y]r_1[s] \text{ comp } c_1$

$T_2$ =insertion compte  $z=400$  et mise à jour  $s=x+y+z=$   $ins_2[z] r_2[s] w_2[s] c_2$

$H= r_1[x]_{x:200} r_1[y]_{y:100} ins_2[z]_{z:400} r_2[x]_{s:300} w_2[s]_{s:700} c_2 r_1[s]_{s:700} \text{ comp }_{x+y:300 \langle s:700} c_1$

Résultat : La vérification dans  $T_1$  trouve une incohérence (objet fantôme z apparu durant  $T_1$ )

Pour éviter cela  $T_1$  devrait mettre un verrou partagé sur toute la table car il est impossible de verrouiller des tuples qui n'existent pas !

Oracle ne prévient pas ce phénomène lorsque le mode est COMMITED READ. Phénomène prévenu dans le niveau REPEATABLE\_READ (ISO).

Oracle offre les transactions READ ONLY et SERIALIZABLE.



# *LES TRANSACTIONS READ ONLY*

Un utilisateur qui exécute une requête SELECT sur une table ou plusieurs tables est sûr de voir toutes les données telles qu'elles étaient au début de l'interrogation, même si d'autres utilisateurs modifient la table et valident leurs modifications pendant ce temps.

- Cette possibilité peut être élargie à une transaction ne comportant que des consultations en débutant la transaction par l'ordre :

**SET TRANSACTION READ ONLY;** (par défaut READ WRITE)

- INSERT, UPDATE, DELETE, SELECT ... FOR UPDATE sont interdits.
- Toute instruction de définition de données (CREATE, ALTER, DROP) met fin à la transaction READ-ONLY.

Ainsi lors des différentes consultations, il n'y a pas de risque d'apparition d'objets fantômes et de lecture non répétitive.

# *LES TRANSACTIONS SERIALISABLE*

- Pour éviter d'obtenir des objets fantômes et de lectures non répétitives à la suite de consultations multiples en vue de mises à jour, il faut utiliser le mode **SERIALIZABLE**.
- Pour permettre ce mode il faut réserver des ressources lors de la création des tables. **CREATE TABLE table(...) INITRANS 3;**
- Oracle commute en mode **SERIALIZABLE** par la commande en début de transaction :

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

Quand une transaction non validée au début de notre transaction sérialisable et ayant modifié des données qui malheureusement ont été consultées par notre transaction , se trouve validée, Oracle génère l'erreur -8177 pour informer l'application que des objets fantômes risquent d'apparaître. Il appartient au programmeur de décider de la démarche à suivre dans ce cas.

# Transaction autonome

- Une transaction autonome fait partie d'une transaction principale.
- Lorsque la transaction autonome se déclenche, la transaction principale est suspendue.
- Dès que la transaction autonome prend fin, la transaction principale reprend.
- Un **COMMIT** peut donc avoir lieu au milieu de plusieurs **ROLLBACK**.

## Quand utiliser une transaction autonome?

- D'un côté, vous voulez journaliser des erreurs dans une table. De l'autre, vous voulez défaire la transaction de la partie principale du fait de l'erreur comme si la transaction toute entière n'était plus atomique.

# AUTONOMOUS TRANSACTION

Une **exception** est levée dans la Main Transaction (MT)

PROCEDURE add\_emp(..) IS

BEGIN

UPDATE..;

MT commence

UPDATE..;

SAVEPOINT start\_add;

INSERT..;

COMMIT;

MT suspendue

EXCEPTION

WHEN OTHERS THEN

ROLLBACK start\_add;

Log (SQLEERM); - - - ->

END;

/

MT s'achève

Autonomous Transaction (AT)

PROCEDURE log(..)

IS

PRAGMA AUTONOMOUS\_TRANSACTION;

BEGIN

AT commence

INSERT INTO log

VALUES (USER, SQLERRM, SYSDATE);

COMMIT;

END;

AT s'achève

/

MT reprend

# *LE GRANULE DE CONCURRENCE*

- Le taux de parallélisme est proportionnel à la finesse du granule de concurrence utilisée. En revanche, le coût (nombre de verrou à poser) lié à la mise en oeuvre est inversement proportionnel à la finesse du granule utilisé.
- Oracle propose par défaut le verrouillage au niveau tuple (niveau fin).
- Pour les mises à jour en masse le niveau relation est plus approprié.

# *VERROUILLAGE EXCLUSIF DE LA TABLE*

Ce mode est activé par l'option EXCLUSIVE (X) de l'ordre LOCK

**LOCK TABLE nom-table IN EXCLUSIVE MODE;**

- Ce verrou est le plus contraignant car il réserve toutes les lignes de la table. Les autres transactions peuvent consulter les données de la table mais ne peuvent ni les modifier, ni poser de verrous sur la table et sur ses lignes.

# *VERROUILLAGE DE LA TABLE EN MODE PARTAGÉ*

Ce mode est activé par l'option SHARE (S) de l'ordre LOCK

**LOCK TABLE nom-table IN SHARE MODE;**

- Lorsqu'une transaction verrouille une table en mode partagé, les autres transactions peuvent mettre différents verrous en mode partagé sur la table (RS ou S), mais aucune autre transaction ne peut demander de verrou de type SRX ou X. Si plusieurs transactions verrouillent une table en mode S, aucune d'elles ne peut effectuer une mise à jour sur les données de la table. Une transaction qui a un verrou SHARE sur une table ne peut donc effectuer une modification de la table que si aucune autre transaction n'a demandé de verrou SHARE.



# *VERROUILLAGE SÉLECTIF DES LIGNES*

Ce mode est activé soit par l'une des options ROW SHARE ou SHARE UPDATE (RS) de l'ordre LOCK, soit par la clause FOR UPDATE de l'ordre SELECT

**LOCK TABLE nom-table IN ROW SHARE MODE;  
SELECT ... FOR UPDATE;**

- Ce verrou est le moins contraignant : il permet aux autres transactions de mettre à jour les données autres que celles concernées par la transaction qui a demandé le verrou. En revanche, aucune autre transaction ne peut demander de verrou de type X.

# *VERROUILLAGE EXCLUSIF DES LIGNES*

Ce mode est activé explicitement par l'option ROW EXCLUSIVE (RX) de l'ordre LOCK ou implicitement lors de l'exécution d'un ordre UPDATE, DELETE ou INSERT.

**LOCK TABLE nom table IN ROW EXCLUSIVE MODE;**

- Il s'agit d'un mode de verrouillage sélectif des lignes un peu plus contraignant que le précédent. Les autres transactions ne peuvent demander ni de verrou en mode partagé (S), ni de verrou exclusif sur la table (X, SRX).

# *VERROUILLAGE EN MODE PARTAGÉ EXCLUSIF DES LIGNES*

Ce mode est activé par l'option SHARE ROW EXCLUSIVE (SRX) de l'ordre LOCK:

**LOCK TABLE nom table  
IN SHARE ROW EXCLUSIVE MODE;**

- Les autres transactions ne peuvent demander aucun verrou exclusif ou partagé sur la table, à l'exclusion du type RS. Une seule transaction peut demander ce type de verrou sur une table.

# *COMPATIBILITÉ DES MODES DE VERROUILLAGE*

Verrou obtenu par une transaction	Pour les autres transactions	
	Verrous impossibles	Verrous possibles
RS	X	RS, RX, S, SRX
RX	S, SRX, X	RX, RS
S	RX, SRX, X	RS, S
SRX	RX, S, SRX, X	RS
X	RS, RX, S, SRX, X	

# *LES PANNES & SGBD*

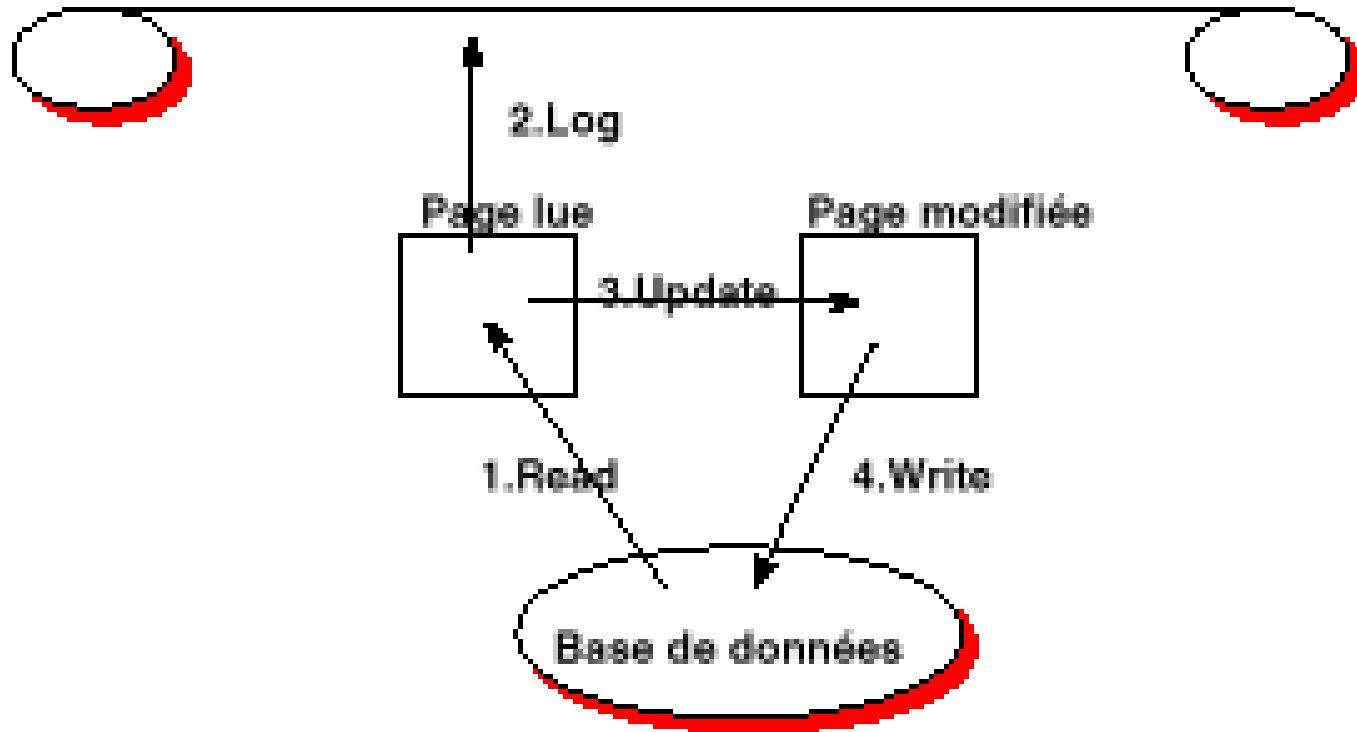
- Une panne de transaction correspond à l'interruption d'une transaction en cours d'exécution. Une telle panne peut se produire dans trois cas:
  - sur décision du programmeur par le biais de l'ordre ROLLBACK.
  - sur décision de l'organe de contrôle de concurrence.
  - suite à une erreur du logiciel d'application.
- Une panne système correspond à une interruption anormale du SGBD.
- Une panne disque correspond à la détérioration de l'espace disque contenant la base de données.

# *MÉCANISMES DE REPRISE*

- Ils garantissent la fiabilité des SGBD. Ils ont le double objectif de faire respecter les propriétés d'atomicité et de durabilité des transactions.
- Suite à une panne, il est nécessaire de défaire ou de refaire les mises à jour d'une ou plusieurs transactions.
  - Défaire les mises à jour d'une transaction nécessite de connaître l'image avant (valeur précédente) de chaque données mises à jour.
  - Refaire les mises à jour d'une transaction nécessite de connaître l'image après (valeur après mise à jour) de chaque données mises à jour.
- Ces informations sont généralement stockées dans des fichiers particuliers appelées respectivement journal des images avant et des images après.

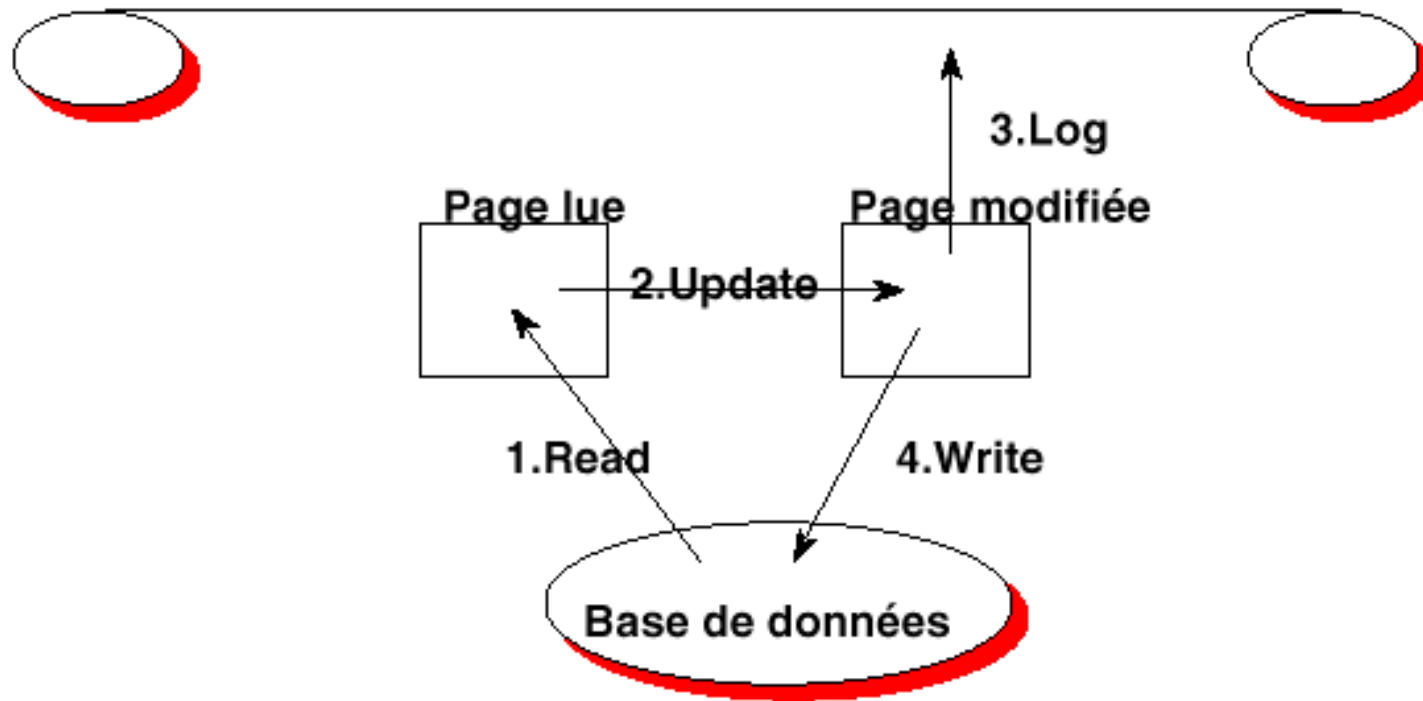
# *JOURNAL DES IMAGES AVANT*

- Utilisé pour défaire les mises à jour
- Toute mise à jour doit être précédée d'une écriture dans le journal des images avant



# *JOURNAL DES IMAGES APRÈS*

- Utilisé pour refaire les mises à jour
- Toute validation de transaction doit être précédée de l'écriture de son journal d'images après sur un disque différent de celui contenant la base de données





# *CONCLUSION : REPRISE AVEC ORACLE*

- Rollback Segment : Stocke l 'Image Avant des données modifiées
- Utilisé pour :
  - les Rollback.
  - La reprise après une panne.
  - La lecture consistante.
- Fichiers de Reprise (Redo Log)

# *NOTION DE ROLLBACK SEGMENTS D 'ORACLE*

- Lorsque vous lancez une instruction LMD pour modifier les données, le processus serveur enregistre d'abord les anciennes valeurs dans un rollback segment. Les rollback segments permettent d'annuler les modifications dans le cas d'une annulation de transaction, et de s'assurer que les autres transactions n'accèdent pas aux modifications non validées.
- Les utilisateurs de la base de données ne peuvent pas accéder ou lire les rollback segments de façon explicite.
- Seul Oracle peut y accéder.

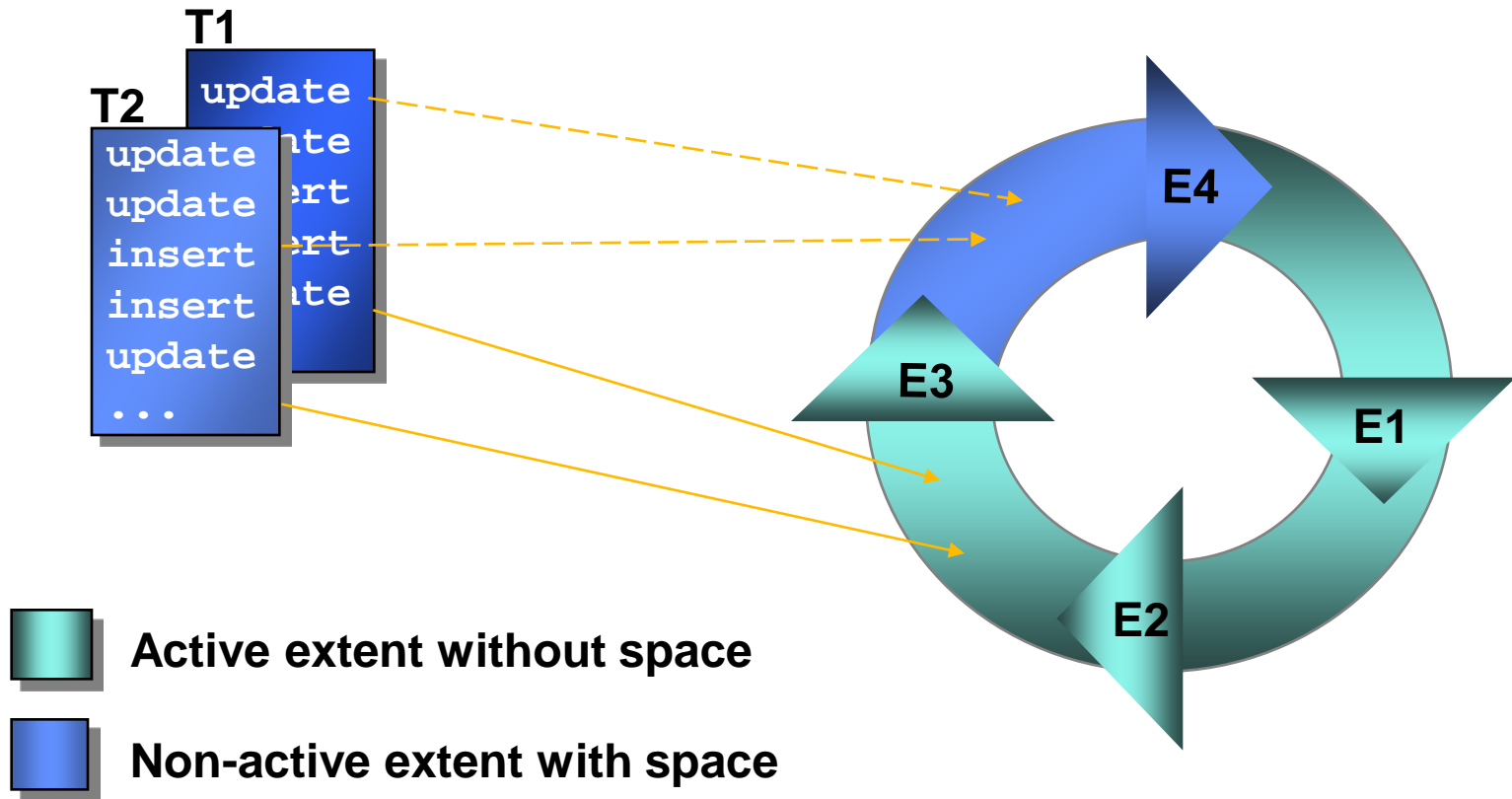
# *AFFECTATION D 'UN ROLLBACK SEGMENT*

- Chaque base de données possède un ou plusieurs rollback segment(s). Oracle affecte automatiquement une transaction au rollback segment suivant disponible.
- L'affectation d'une transaction au rollback segment se produit lorsque vous lancez la première instruction LMD de la transaction.
- Oracle n'affecte pas de transactions en lecture seule à un rollback segment Les transactions en lecture seule sont celles qui ne contiennent que des requêtes (Select).

# *UTILISATION DES ROLLBACK SEGMENTS*

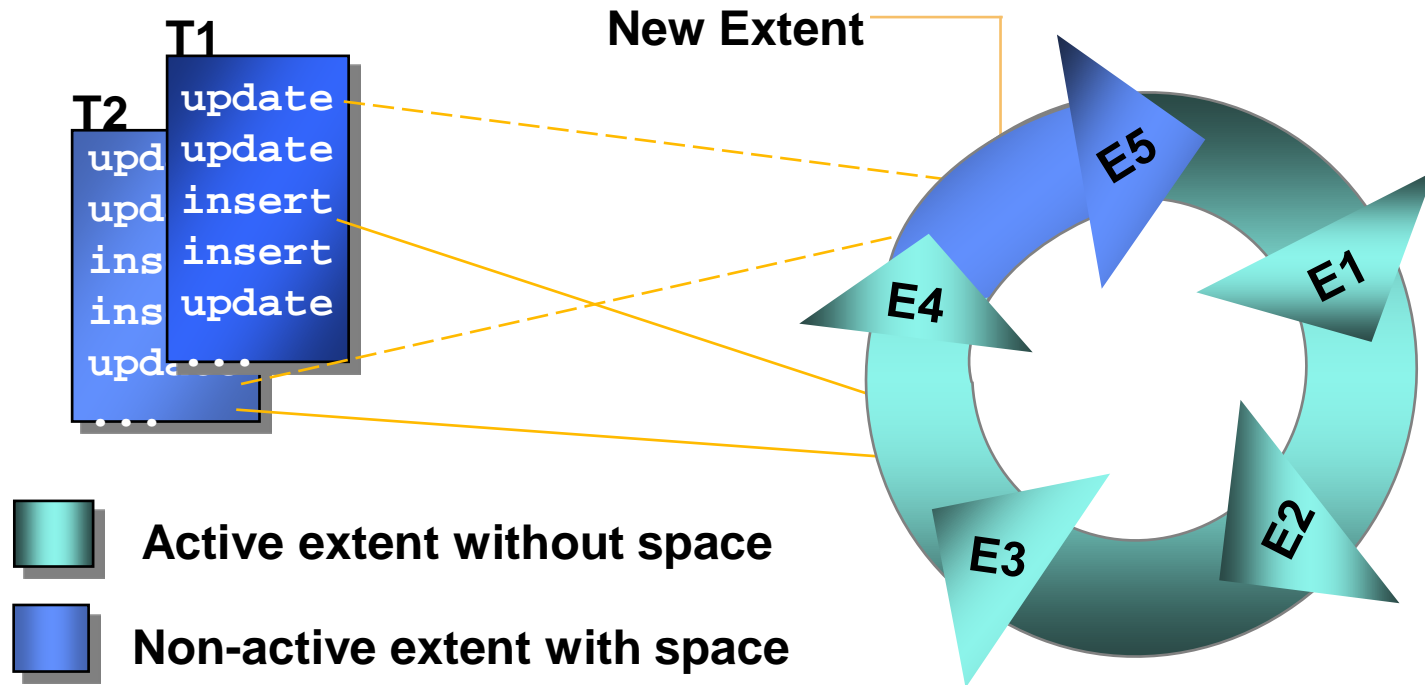
- Si nécessaire, les rollback segments permettent d'annuler les transactions et ainsi les modifications qui ont été apportées aux données.
- Les informations stockées dans les rollback segments sont aussi utilisées pour la lecture cohérente. Cela permet de s'assurer que les autres transactions n'accèdent pas aux modifications non validées effectuées par une instruction LMD.
- En outre, les rollback segments sont utilisés pour restaurer la base de données dans un état cohérent en cas de pannes diverses.

# *LA GESTION DES ROLLBACK SEGMENTS*



Les transactions se déroulent normalement tant qu'il reste de la place ...

# EXTENSION D 'UN ROLLBACK SEGMENT



Seul le BDA peut le faire.

Tout ce qu'un autre utilisateur peut faire, c'est d'écourter ses transactions quand cela est possible. Par exemple, `SET AUTOCOMMIT=100` provoque la validation des transactions tous les 100 ordres exécutés.

# *REDO LOG*

- Ils contiennent tous les changements d'états de la base de données et sont utiles pour la restauration d'une base suite à un incident d'instance ou de disque.
- Cette restauration consiste en une ré-application du contenu des fichiers redo log sur la base.
- Ces fichiers de journalisation des modifications successives sont au moins au nombre de deux. Lorsque le fichier redo log courant est plein, Oracle enchaîne sur le suivant, et ainsi de suite jusqu'au dernier. Quand celui-ci est plein, Oracle fait le tour et réutilise le premier, puis le second. L'utilisation de ces fichiers est donc circulaire.

# *CONTENU DU REDO LOG*

Un fichier Redo Log est rempli séquentiellement par des enregistrements contenant les valeurs de colonnes de tables avant et après modification.

Le contenu d'un enregistrement dans un fichier Redo Log est le suivant :

- numéro de transaction,
- date de transaction,
- état transaction,
- adresse colonne,
- valeur colonne avant modification,
- valeur colonne après modification.



# *MODE D'UTILISATION*

Les fichiers redo log peuvent être utilisés :

- soit de façon unique (par défaut),
- soit de façon multiplexée.

## *FAÇON UNIQUE*

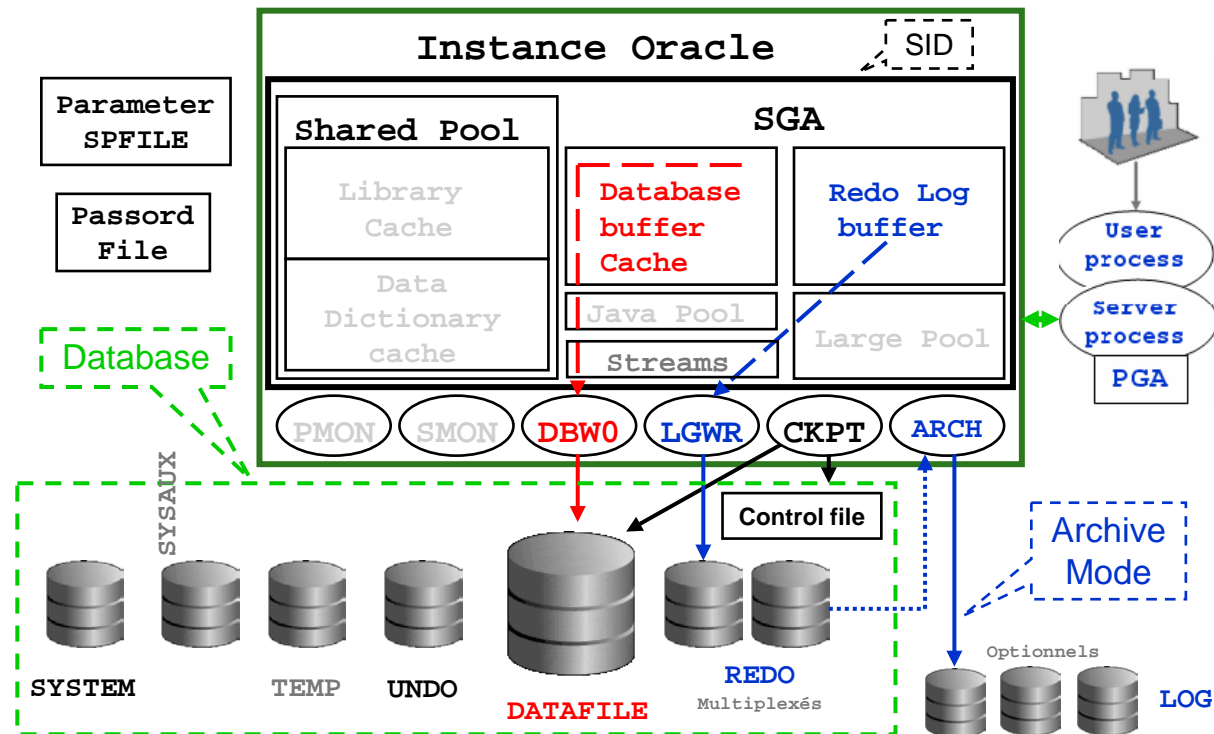
L'utilisation de façon unique implique les règles suivantes :

- un seul fichier Redo Log est en service à un moment donné,
- quand un fichier Redo Log est plein, le deuxième fichier est alors mis en service,
- les modifications ne sont stockées qu'une seule fois.

# *FAÇON MULTIPLEXÉE*

- Comme des fichiers de reprise peuvent être eux aussi altérés, oracle permet d'avoir des fichiers de reprises multiples. Il s'agit d'une duplication des fichiers reprises sur différents disques. Lorsque l'un de ces fichiers est perdu ou endommagé, son image sur l'autre disque peut être utilisée.
- L'ensemble de fichiers reprise qui sont actifs simultanément sont constituent un groupe. Chaque fichier de ce groupe est dit membre.
  - chaque groupe de fichiers Redo Log contient les mêmes informations,
  - Chaque membre d'un groupe est mis à jour simultanément,

# DEROULEMENT D'UNE TRANSACTION ORACLE



1. Oracle examine le cache de tampons pour déterminer si le bloc de données cible se trouve en mémoire.
2. Si le bloc de données cible ne se trouve pas en mémoire, Oracle le récupère à partir du disque.
3. Il enregistre une entrée de reprise dans le tampon REDO.
4. Il consigne dans un segment d'annulation, le code et les données requises pour défaire la modification apportée au bloc (BEFORE IMAGE).  
Auparavant, des vecteurs de changement sont générés pour cette action.
5. Oracle met à jour le bloc de données en mémoire avec la nouvelle valeur
6. Oracle génère une entrée de validation dans le tampon REDO et associe à la transaction un SCN de validation.
7. Il écrit dans le journal de reprise sur disque le contenu du tampon REDO.
8. Il libère les blocs du segment d'annulation (BEFORE IMAGE) qui contenaient les informations d'annulation de la transaction.
9. Il enregistre le bloc modifié dans un fichier de données.



# Chapitre 13

## Les passerelles SGBD

### JDBC

Michel Dubois

*Michel.Dubois@univ-ubs.fr*

[Retour au plan](#)

# Préalable à la programmation SGBDR

Disposer d'une base de donnée relationnelle ayant les propriétés suivantes :

Un schéma conceptuel conforme à la réalité (Analyse, règles de gestion).

Un schéma logique permettant la cohérence des données:

- Un schéma logique normalisé.

- Un schéma logique complet. (contraintes de domaines, contraintes de clés, intégrité référentielle, obligation, unicité, de valeurs, plus complexes ).

Un schéma physique optimisé. (ajout d'attributs calculés, création d'index).

# Programmation avec SGBD

- Valeurs nulles.
- Notion de concurrence.
- On sépare clairement l'acquisition des données de leur traitement.

## **Interrogation : SELECT FROM WHERE**

- Renvoie un ensemble de tuple (vide, singleton ou autre).
- Utilisation possible de curseurs SQL.

## **Mise à jour : CREATE, ALTER, DROP, INSERT, DELETE, UPDATE**

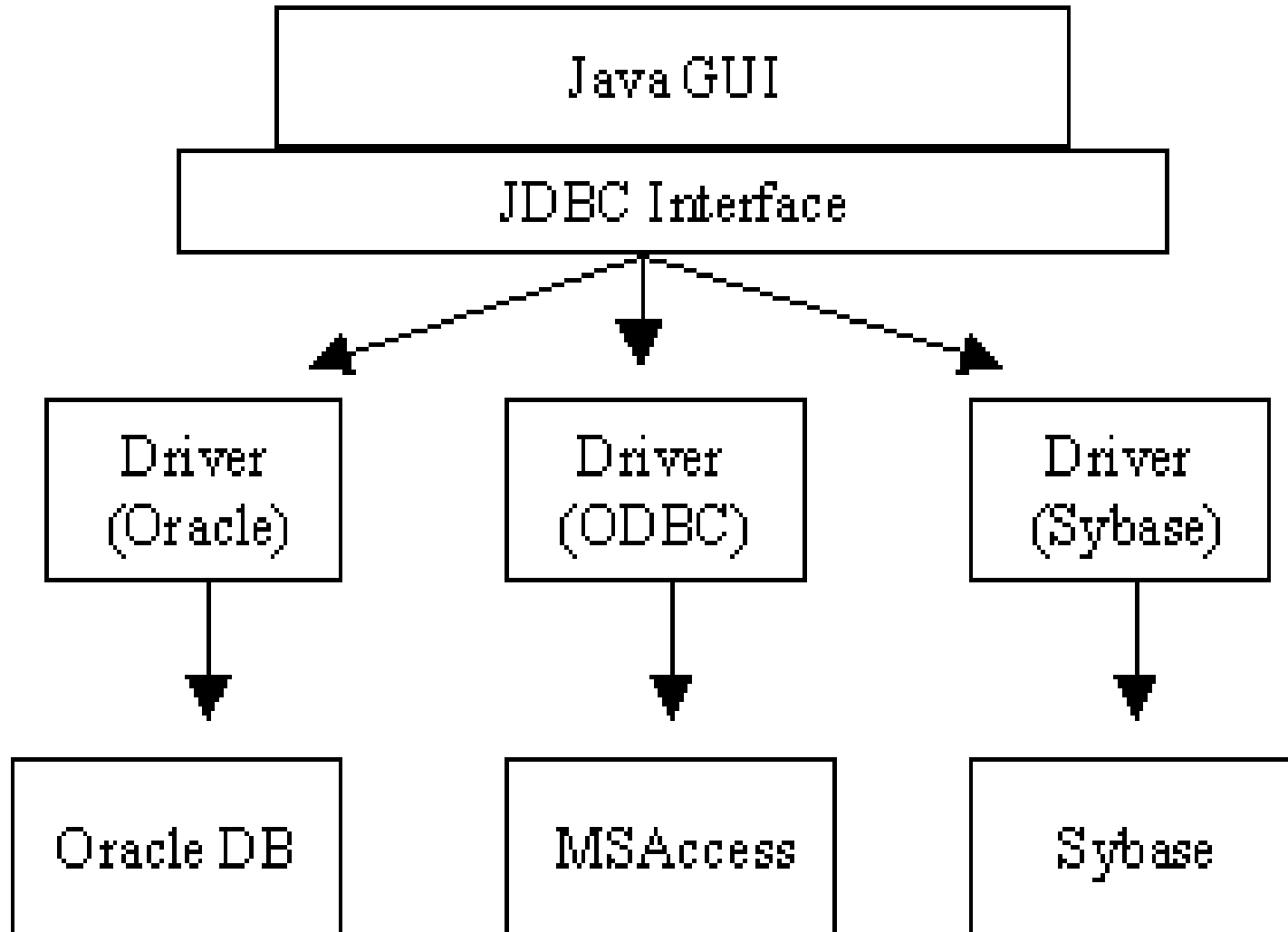
- Notion de transaction très importante.
- Notion de vue non modifiable
- Ne renvoie pas de résultat à part un message d'erreur.

# Objectifs

- Fournir un accès homogène aux SGBDR
- Abstraction des SGBDR cibles
- Requêtes SQL
- Simple à mettre en oeuvre
- *Core* API (JDK 1.1 et JDBC 1.0)
- *Core* API et *extension* API (JDK 1.2 et JDBC 2.0)



# Le principe de JDBC



# Plan

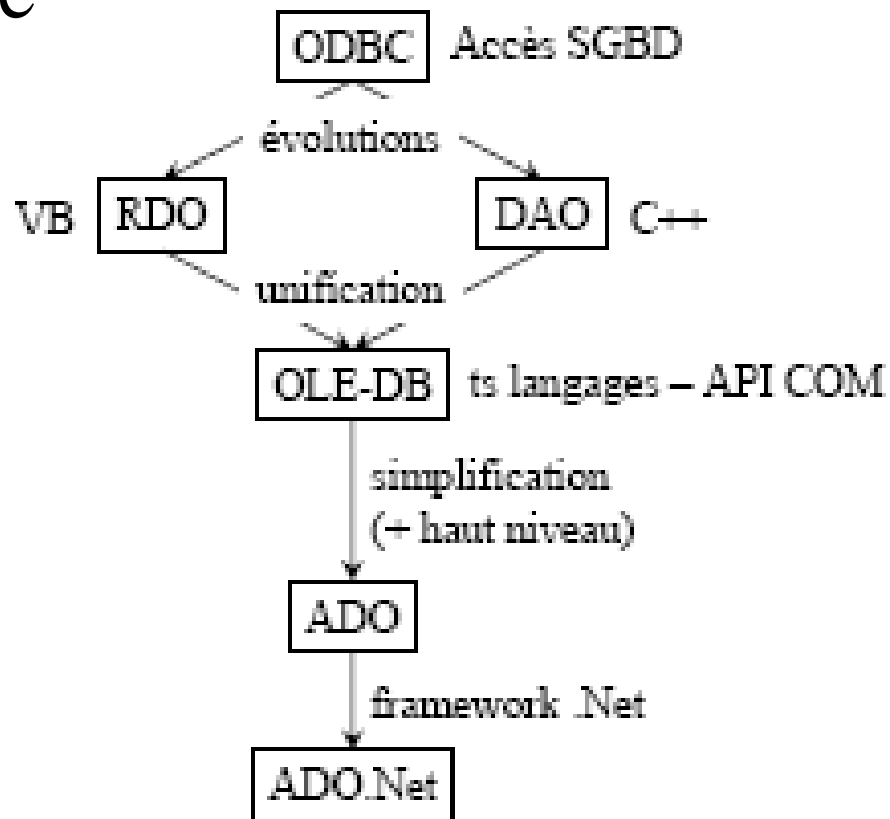
- Mise en oeuvre avec les pilotes JDBC
- L ' API JDBC et son utilisation
  - Basique et avancée
  - Pilote interne Oracle dans JVM Aurora
  - Objet-relationnel et JDBC
- ORM avec un exemple commenté : la Fnuc

# Les pilotes JDBC

- ODBC
- Types de pilotes JDBC
- Exemples de pilotes JDBC

# ODBC et ADO .Net

- Les API MS d'interrogation SGBD
- Petit historique



# Les types de pilotes JDBC

- JDBC interagit avec le SGBDR par un *driver*
- Il existe des *drivers* pour Oracle, Sybase, Informix, DB2, ...
- 4 types de drivers :
  - 1. *Bridge* ODBC (fourni avec JDBC)
  - 2. *Native-API partly-Java driver*
  - 3. *JDBC-Net all-Java driver*
  - 4. *Native-protocol all-Java driver*
- 1. et 2. nécessite des architectures 3-tiers pour les applets

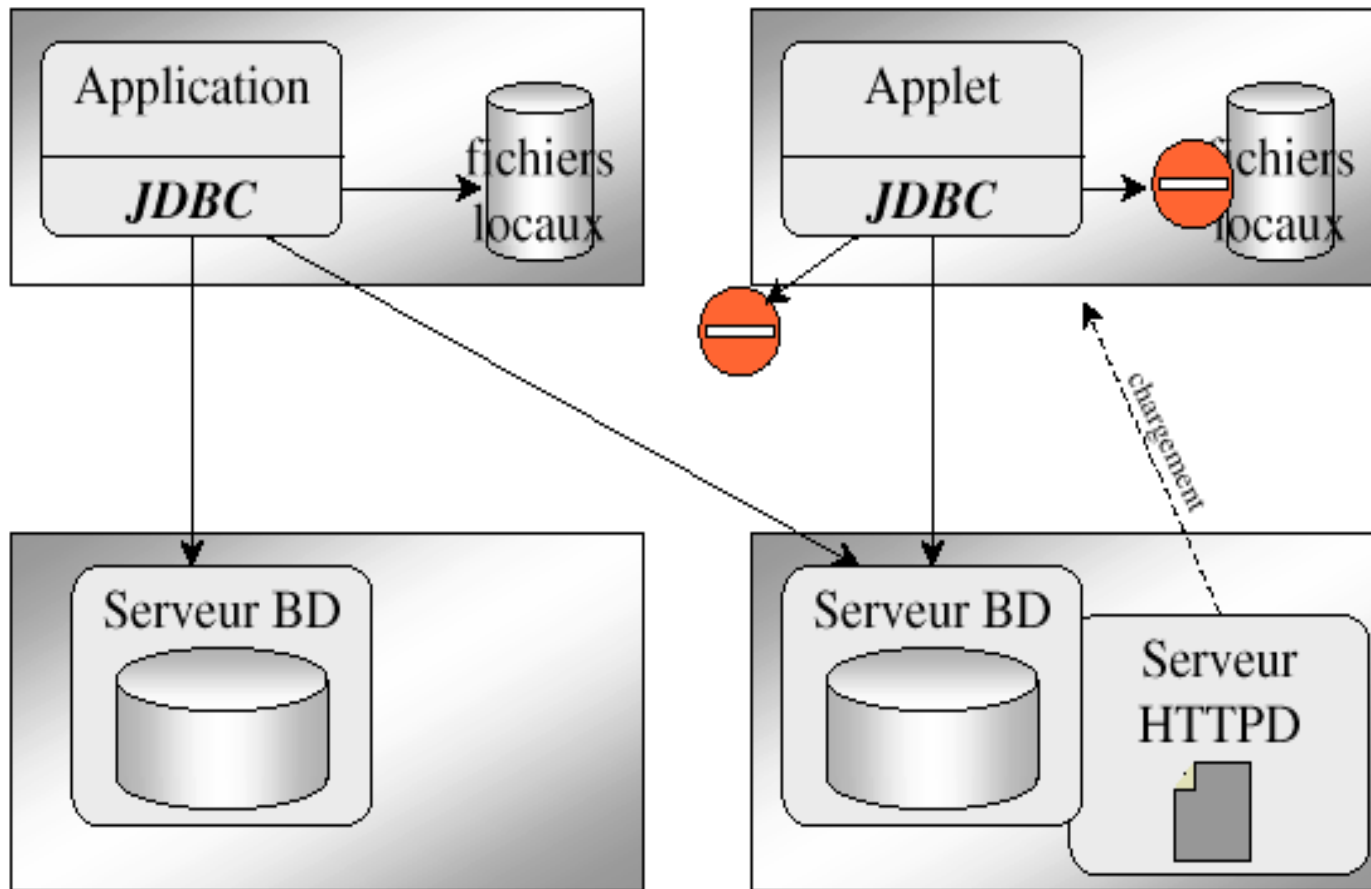
# Modèle de sécurité Java (avant 1.2)

- Pour les application Java, il n'y a aucune restriction a priori. Il est possible alors de définir ses propres règles, en créant une sous classe de la classe abstraite `SecurityManager` et en l'associant à l'application par la méthode de la classe `System.setSecurityManager`.
- La sécurité générique s'applique de toute manière :
  - sécurité du langage lui-même (des références au lieu des pointeurs, la déclaration d'une classe ou d'une méthode finales)
  - vérification du bytecode téléchargé par la machine virtuelle
  - Le chargement dynamique des classes fait la distinction entre les classes locales et les classes téléchargés

# Modèle de sécurité Java (1.1)

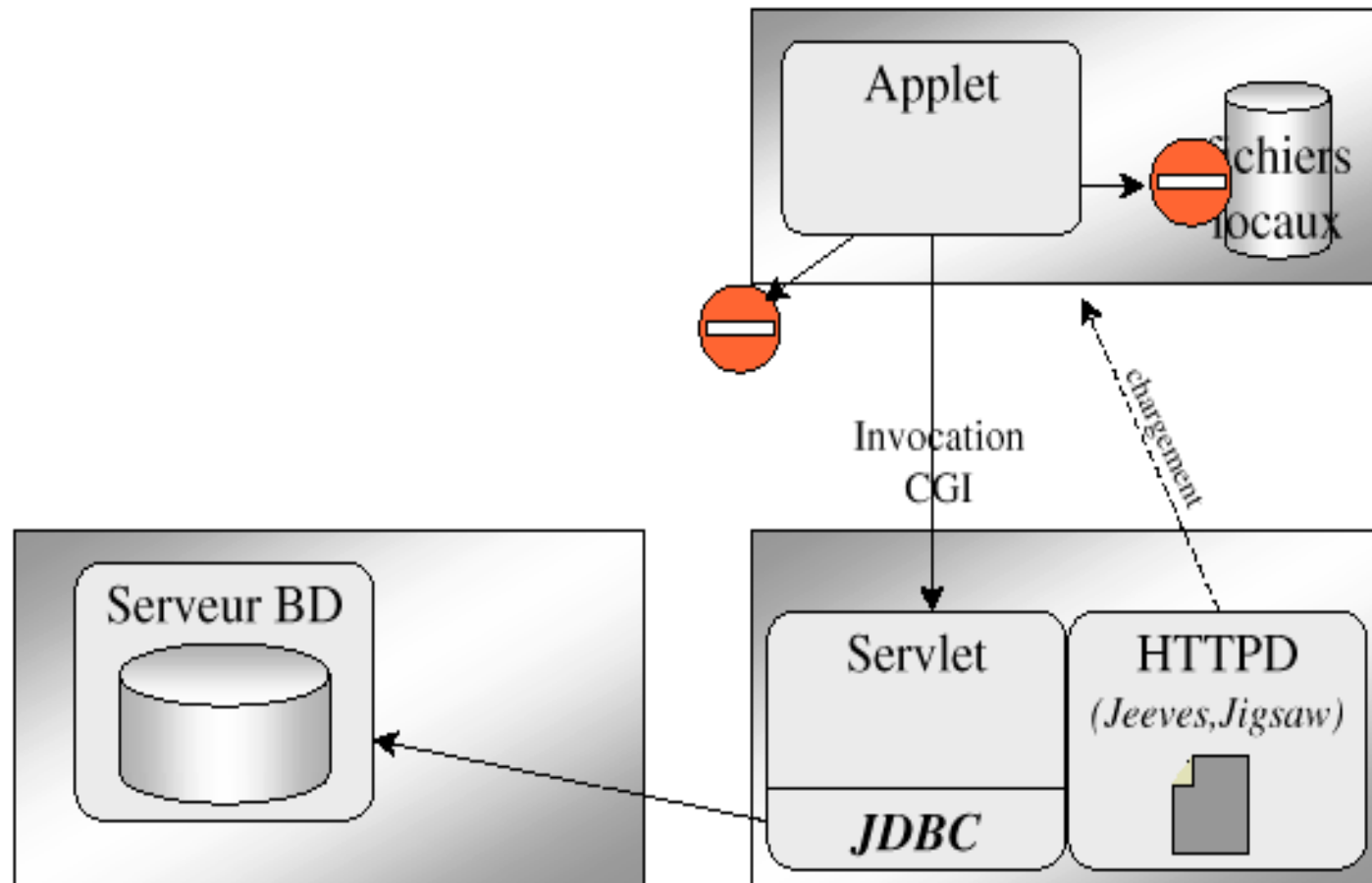
- Modèle classique pour les applets (modèle par défaut des navigateurs) : l'applet est soumise au *Security Manager* qui permet au navigateur les restrictions suivantes :
  - pas d'accès en lecture ni en écriture sur le disque du browser.
  - connexion réseau uniquement sur le serveur d'origine.
  - pas de chargement de librairie native.
  - pas de lancement de processus, ...
- Cependant, une applet digne de confiance (trusted) ou applet signées se voit autoriser l'accès à ces ressources. Le code téléchargé est accompagné d'un certificat garantissant son origine et il est sécurisé par un système de cryptage pour le transfert.

# JDBC, Applets untrusted et Applications





# JDBC, Applet untrusted, Servlet



# Modèle de sécurité Java 1.2

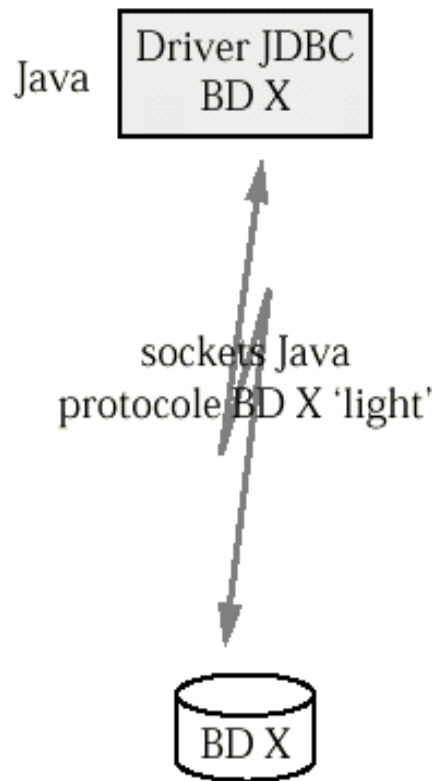
- Java 2 a introduit une nouvelle architecture de sécurité qui permet d'étendre les actions autorisées aux applets avec des fichiers de police de sécurité ou/et en signant les applets.
- On peut espérer que les prochaines versions des navigateurs adopteront tous la nouvelle architecture de sécurité de Java 2.
- En attendant, il est possible de charger sur les navigateurs le plug-in Java fourni par Sun avec le SDK. Ce plug-in est disponible sur la machine locale si le runtime java y est installé ; sur certains systèmes il faut en plus positionner des variables d'environnements.

# Modèle de sécurité Java 1.2

- Plus de distinction code local, distant. Tout type de programme java (Application, Applet, JavaBeans, servlets) peut faire l'objet de restrictions.
- On peut affiner les permissions que l'on donne au programme JAVA. Cette possibilité existait déjà en programmant les classloader et le gestionnaire de sécurité. Maintenant la politique de sécurité est facilement configurable.
- Cette politique de sécurité déclarative fait la correspondance entre les caractéristiques du code (URL, clés publiques de certification) et des droits d'accès (cible, actions) contenus dans des classes (FilePermission, SocketPermission, AWTPermission, RuntimePermission).

# Driver JDBC Type 4

## Native Protocol All-Java Driver



- **Client 'fin'**
  - Driver traduit les appels JDBC en appels à l'API du SGBD
  - Driver crée des sockets pour émuler le protocole BD X 'light'
- **Avantage / Désavantage**
  - Tout Java = portable
  - Tout Java = charger Driver (~300K)
- **Utilisation**
  - Applets en architecture 2-tier

# Plan

- Mise en œuvre avec les pilotes JDBC
- L ' API JDBC et son utilisation
  - Basique et avancée
    - Pilote interne Oracle dans JVM Aurora
    - Objet-relationnel et JDBC
- ORM avec un exemple commenté : la Fnuc

# API JDBC

## **Interface**

Driver

Connection

DatabaseMetaData

Statement

PreparedStatement

CallableStatement

ResultSet

ResultSetMetaData

# API JDBC

## **Classes :**

Date

DriverManager

DriverPropertyInfo

Time

Timestamp

Types

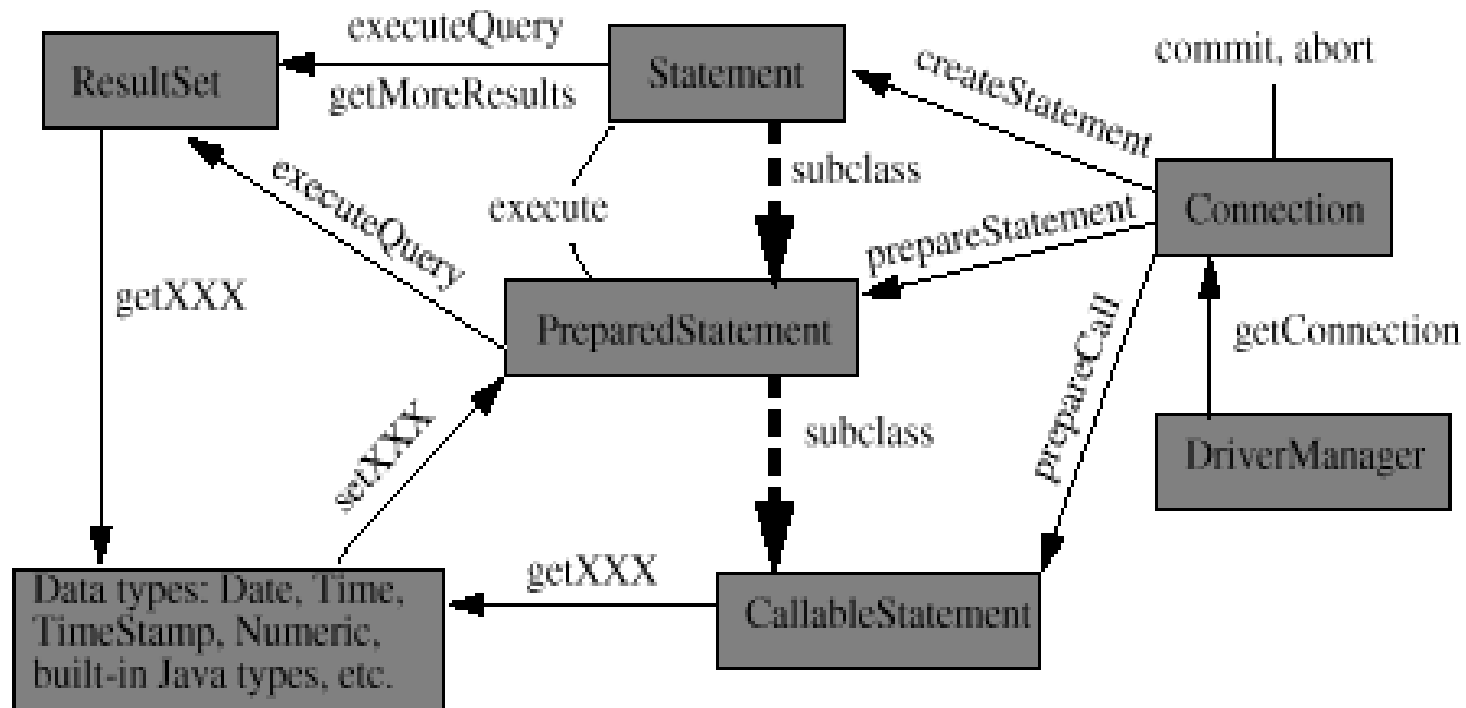
## **Exception :**

DataTruncation

SQLException

SQLWarning

# Utilisation de l'API





# Utilisation de l'API

- Charger le *driver*
- Connexion à la base
- Création d'un *statement*
- Exécution de la requête
- Lecture des résultats

# Chargement du *driver*

- Utiliser la méthode `forName` de la classe `Class` :
  - `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
  - `Class.forName("org.postgresql.Driver");`
  - `Class.forName("oracle.jdbc.driver.OracleDriver");`
  - `Class.forName("org.apache.derby.jdbc.EmbeddedDriver");`
  - `Class.forName("com.sas.net.sharenet.ShareNetDriver");`
  - `Class.forName("interbase.interclient.Driver");`
- Inutile avec JDBC 4 !

# Connexion à la base

- Accès via un URL qui spécifie :
  - l'utilisation de JDBC
  - le driver ou le type du SGBDR
  - l'identification de la base
- Exemple :
  - `jdbc:odbc:mabase`
  - `jdbc:postgresql://hostname:port/mabase`
  - `jdbc:oracle:thin:@hostname:port:SID`
  - `jdbc:interbase://hostname//interbase/data/em.gdb`
- Extension JDBC3.0 aux fichiers plats et aux feuilles de calcul
- Ouverture de la connexion :

```
Connection conn = DriverManager.getConnection(url, user, password);
```

Michel Dubois 2011-2012

# Connexion à la base : URL complète

- Accès via un URL qui spécifie :
  - l'utilisation de JDBC
  - le driver ou le type du SGBDR
  - l'identification de la base :
    - hostname : nom de la machine hébergeant le serveur de données
    - port : port d'écoute du serveur de données
    - mabase : nom de la base de données
    - SID : instance de la base de données Oracle
  - l'identification de l'utilisateur
    - l : login
    - m : mot de passe
- Exemple :
  - jdbc:postgresql://hostname:port/mabase?user=l&password=m
  - jdbc:postgresql://hostname:port/mabase?user=l&#38;password=m (fichier XML)
  - jdbc:mysql://hostname:port/mabase?user=l&password=m
  - jdbc:mysql://hostname:port/mabase?user=l&#38;password=m (fichier XML)
  - jdbc:oracle:thin:l/m@hostname:port:SID
- Ouverture de la connexion :

```
Connection conn = DriverManager.getConnection(url);
```

# Connexion aux bases de l'I.U.T.

Vous pouvez accéder à quatre SGBD (+dataset SAS) à l'aide des pilotes de type 4.

- PostgreSQL 8.3 sur wamba :
  - pilote : "org.postgresql.Driver"
  - URL : "jdbc:postgresql://wamba.univ-ubs.fr:5432/csdi3"
  - Utilisateur : "csdi3", mot de passe : "csdi344"
- Oracle 10g sur oraetud (hors radius) :
  - pilote : "oracle.jdbc.driver.OracleDriver"
  - URL : "jdbc:oracle:thin:@oraetud.univ-ubs.fr:1521:ORAETUD"
  - Utilisateur : mdubois, mot de passe : michel
- MySQL sur wamba :
  - pilote : "com.mysql.jdbc.Driver"
  - URL : "jdbc:mysql://wamba.univ-ubs.fr/p\_07\_dubois\_fnuc"
  - Utilisateur : "p\_dubois\_view", mot de passe : "toto"
- Apache Derby embarqué :
  - pilote : "org.apache.derby.jdbc.EmbeddedDriver"
  - URL : "jdbc:derby:classpath:/fnucdb"
  - Utilisateur : "APP", mot de passe : "APP"
- Serveur SAS SHARE 9.2 sur saskatchewan :
  - pilote : "com.sas.net.sharenet.ShareNetDriver "
  - URL : "jdbc:sharenet://saskatchewan.univ-ubs.fr:8551"
  - Utilisateur : "eleve", mot de passe : "eleve"

# JDBC & Oracle 10g avec radius

- Si accès à ORAETUD

ORACLE\_HOME=...

```
CLASSPATH=$CLASSPATH:$ORACLE_HOME/jdbc/lib/ojdbc14.jar:$ORACLE_HOME/jdbc/lib/orai18n.jar:$ORACLE_HOME/md/lib/ocrs12.zip:$ORACLE_HOME/md/lib/sdoapi.jar:$ORACLE_HOME/md/lib/sdovis.jar:$ORACLE_HOME/md/lib/sdonm.jar:$ORACLE_HOME/md/lib/sdoutl.jar
```

- Sinon

ORALIB=...

```
CLASSPATH=$CLASSPATH:$ORALIB/ojdbc14.jar:$ORALIB/orai18n.jar:$ORALIB/ocrs12.zip:$ORALIB/sdoapi.jar:$ORALIB/sdovis.jar:$ORALIB/sdonm.jar:$ORALIB/sdoutl.jar
```

Le pilote JDBC THIN 10g ne supporte pas l'authentification radius contrairement au pilote OCI et aux pilotes 11g OCI/THIN.

- sdo\*.jar extension SDO de jdbc
- ocrs12.zip permet d'étendre les ResultSet

```
import java.sql.*;
import oracle.sql.STRUCT;
import oracle.jdbc.driver.*;
import oracle.jdbc.OracleConnection;
import oracle.spatial.geometry.JGeometry;
```

```
Class.forName("oracle.jdbc.driver.OracleDriver");
static final String url = "jdbc:oracle:thin:@oraetud.univ-ubs.fr:1521:ORAETUD";
Connection conn = DriverManager.getConnection(url,"mdubois","michel");
```

A l'UBS, il faut utiliser le pilote OCI, le serveur Oracle 10g utilisant l'authentification radius :

```
java.util.Properties props = new
    java.util.Properties();
props.setProperty(
    "sqlnet.authentication.services",
    "radius");
props.setProperty("user","e0602180");
props.setProperty("password","1234");
String url="jdbc:oracle:oci:"
    +"@oraetud.univ-ubs.fr:1521:ORAETUD";
conn =
```

```
DriverManager.getConnection(url,props);
```

# JDBC & Oracle 11g (Oracle 10g avec radius)

- Si accès à ORAETUD avec un JDK 1.6 : `ojdbc6.jar` sinon `ojdbc5.jar` pour le JDK 1.5  
ORACLE\_HOME=...  
CLASSPATH=\$CLASSPATH:.:\$ORACLE\_HOME/jdbc/lib/ojdbc16.jar:\$ORACLE\_HOME/jdbc/lib/orai18n.jar:\$ORACLE\_HOME/md/lib/sdoapi.jar:\$ORACLE\_HOME/md/lib/sdovis.jar:\$ORACLE\_HOME/md/lib/sdonm.jar:\$ORACLE\_HOME/md/lib/sdoutl.jar
- Sinon  
ORALIB=...  
CLASSPATH=\$CLASSPATH:.:\$ORALIB/ojdbc16.jar:\$ORALIB/orai18n.jar:\$ORALIB/sdoapi.jar:\$ORALIB/sdovis.jar:\$ORALIB/sdonm.jar:\$ORALIB/sdoutl.jar

Le pilote JDBC THIN 11g supporte l'authentification radius.

- `sdo*.jar` extension SDO de jdbc
  - `ocrs12.zip` permet d'étendre les ResultSet
- ```
import java.sql.*;
import oracle.sql.STRUCT;
import oracle.jdbc.driver.*;
import oracle.jdbc.OracleConnection;
import oracle.spatial.geometry.JGeometry;
```

```
Class.forName("oracle.jdbc.driver.OracleDriver");
static final String url = "jdbc:oracle:thin:@oraetud.univ-ubs.fr:1521:ORAETUD";
Connection conn = DriverManager.getConnection(url,"mdubois","michel");
```

A l'UBS, on peut utiliser le pilote THIN, le serveur Oracle 10g utilisant l'authentification radius :

```
java.util.Properties props = new
    java.util.Properties();
props.setProperty(
    "sqlnet.authentication.services",
    "radius");
props.setProperty("user","e0602180");
props.setProperty("password","1234");
String url="jdbc:oracle:thin:"
    +"@oraetud.univ-ubs.fr:1521:ORAETUD";
conn =
    DriverManager.getConnection(url,props);
```

# Environnement JDBC Oracle

```
@echo off
rem lib/setJDBC.bat
rem Script d'environnement du pilote JDBC Oracle 11g
set basedir=C:\temp\lib
set JAVA_HOME=C:\Program Files\Java\jdk1.6.0_20
if exist "%JAVA_HOME%" goto javaHomeOk
echo JAVA_HOME (%JAVA_HOME%) does not exist
goto end
:javaHomeOk
set ORACLE_BASE= C:\oracle
echo Oracle: %ORACLE_BASE%
echo Using JAVA_HOME %JAVA_HOME%
set PATH=%JAVA_HOME%\bin;%PATH%
set PATH=%basedir%\frameworks\embedded\bin;%PATH%
set CLASSPATH=.;%basedir%\ojdbc6.jar;%basedir%\orail8n.jar
echo Using Classpath %CLASSPATH%
:end
```

```
#!/bin/tcsh
# lib/setJDBC.tcsh
set basedir=/ubs/projets/e_10_lpue4cgi_00/private_html
setenv ORACLE_BASE /vol/app/oracle11g
setenv ORACLE_HOME /vol/app/oracle11g/home
echo " Oracle: '${ORACLE_BASE}'"
if ( ${?CLASSPATH} ) then
    setenv CLASSPATH
    ${basedir}/lib/ojdbc6.jar:${basedir}/lib/orail8n.jar::
    ${CLASSPATH}
else
    setenv CLASSPATH
    ${basedir}/lib/ojdbc6.jar:${basedir}/lib/orail8n.jar:.
endif
echo " ClassPath: '${CLASSPATH}'"
if ( ${?LD_LIBRARY_PATH} ) then
    setenv LD_LIBRARY_PATH
    /vol/app/oracle11g/home/lib:${LD_LIBRARY_PATH}
else
    setenv LD_LIBRARY_PATH /vol/app/oracle11g/home/lib
endif
echo " Bibliothèques OCI: '${LD_LIBRARY_PATH}'"
```

```
#!/bin/bash
# lib/setJDBC.bash
# Script d'environnement du pilote JDBC Oracle 11g
#
#HISTORY:
# 2010/12/08 - MD : Creation
basedir=/ubs/projets/e_10_lpue4cgi_00/private_html
# Pour l'OCI sur distribution Debian locale
export ORACLE_BASE=/vol/app/oracle11g
export ORACLE_HOME=/vol/app/oracle11g/home
echo " Oracle: $ORACLE_BASE"
if [ -z ${CLASSPATH} ]; then
export
    CLASSPATH="${basedir}/lib/ojdbc6.jar:${basedir}/
    lib/orail8n.jar::${CLASSPATH}"
else
export
    CLASSPATH="${basedir}/lib/ojdbc6.jar:${basedir}/
    lib/orail8n.jar:."
fi
echo " ClassPath: $CLASSPATH"
if [ -z ${LD_LIBRARY_PATH} ]; then
export
    LD_LIBRARY_PATH="/vol/app/oracle11g/home/lib:${L
    D_LIBRARY_PATH}"
else
export LD_LIBRARY_PATH="/vol/app/oracle11g/home/lib"
fi
echo " Bibliothèques OCI: $LD_LIBRARY_PATH"
```



# JDBC, Oracle avec radius & Talend

- La connexion hors radius est fonctionnelle pour le composant `tOracleConnection`.

**OracleConnection\_1**

**Paramètres simples**

Type de propriété: Built-In

Paramètres avancés

Type de connexion: Oracle SID

Paramètres dynamiques

Version de la base de données: Oracle 10

Vue

☐ Utilisez un fichier TNS

Documentation

Hôte: "oraetud.univ-ubs.fr" Port: "1521"

Base de données: "ORAETUD" Schéma: "ROOT"

Utilisateur: "mdubois" Mot de passe: "michel"

Paramètres JDBC additionnels: ""

\*Note: Example for Additional JDBC Parameters: "parameterName1=value1&parameterName2=value2"

☐ Utiliser ou enregistrer une connexion partagée à une base de données

- La connexion radius n'est pas durable et robuste pour le composant `tOracleConnection`.
- Pour chaque tâche concernant Oracle, il faut se reconnecter et donc préciser les paramètres de connexion (Oracle SID, Oracle 11) y compris `oracle.net.authentication_services=(RADIUS)`.
- Ex: pour le composant `tOracleOutput`.

**tOracleOutput\_2**

**Paramètres simples**

Type de propriété: Built-In

☐ Utiliser une connexion existante

Type de connexion: Oracle SID

Version de la base de données: Oracle 11

Hôte: "oraetud.univ-ubs.fr" Port: "1521"

Base de données: "ORAETUD" Schéma Oracle: ""

Utilisateur: "e0000000" Mot de passe: [REDACTED]

Table: "NEBRASKA"

Action sur la table: Supprimer la table si elle existe et la créer Action sur les données: Insert

Schéma: Built-In Editer le schéma Synchroniser les colonnes

☐ Terminer en cas d'erreur

**tOracleOutput\_2**

Paramètres simples

**Paramètres avancés**

Paramètres dynamiques

Vue

Documentation

Paramètres JDBC additionnels: "oracle.net.authentication\_services=(RADIUS)"

\*Note: Example for Additional JDBC Parameters: "parameterName1=value1&parameterName2=value2"

Commencer toutes les: 10000

☐ Statistiques du tStatCatcher

Colon

# Site du projet avec la technologie java web (servlets, JSP) sur wamba.univ-ubs.fr

- Le client Oracle n'est pas installé. Il est rappelé que donc la partie oci du pilote JDBC d'Oracle n'est pas disponible. Il faut utiliser le pilote JDBC pour Oracle 11g qui permet l'authentification radius aussi avec la partie thin du pilote; ce qui n'est pas le cas pour le pilote JDBC Oracle 10 g. A noter que c'est le JDK 1.6 qui est utilisé avec Apache Tomcat 5.5 lié à Apache par le mod\_jk (directement adressable via le port 80) donc c'est ojdbc6.jar qui est utilisé par se connecter à Oracle et qui se trouve dans WEB-INF/lib.
- Le répertoire /ubs/projets/<<nomDuProjet>>/private\_html est monté en S:\<<nomDuProjet>> avec gestion des droits automatique (o+r pour fichier, o+rx pour sous répertoire) pour la publication web pour les projets web. Bien entendu, il faut passer par S: (site) plutôt que par P: (projet).
- L'application java web doit se trouver dans le sous répertoire S:\<<nomDuProjet>>\webapps qui contient WEB-INF.
- Les requêtes autorisées sont celles provenant de :

<http://www-i.univ-ubs.fr/etud/projets/<<nomDuProjet>>/>

# Création d'un *Statement*

- 3 types de *statement* :
  - `statement` : requêtes simples
  - `prepared statement` : requêtes pré-compilées
  - `callable statement` : procédures stockées
- Création d'un *statement* :

```
Statement stmt = conn.createStatement();
```

# Execution d'une requête

- 3 types d'executions :
  - `executeQuery` : pour les requêtes qui retournent un `ResultSet`
  - `executeUpdate` : pour les requêtes `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE` et `DROP TABLE`. Renvoie le nombre de lignes mises à jour (int).
  - `execute` : méthode générique qui permet de traiter aussi bien les interrogations que les mises à jour. Renvoie un boolean qui est vrai si le résultat est un ou plusieurs `ResultSet` (`getResultset()`). Sinon il s'agit d'un count (cas d'une mise à jour) : `getUpdateCount()`. Utile pour le SQL dynamique.

# Execution d'une requête d'interrogation

- Execution de la requête :

```
String myQuery = "SELECT prenom, nom, email " +  
                 "FROM employe " +  
                 "WHERE (nom='Dupont') AND (email IS NOT NULL) " +  
                 "ORDER BY nom";
```

```
ResultSet rs = stmt.executeQuery(myQuery);
```

# Lecture des résultats de l'interrogation (1)

- `executeQuery()` renvoie un `ResultSet`
- Le RS se parcourt itérativement *ligne* par *ligne*
- Les colonnes sont référencées par leur numéro ou par leur nom
- L'accès aux valeurs des colonnes se fait par les méthodes `getXXX()` où `xxx` représente le type de l'objet

# Lecture des résultats de l'interrogation (2)

```
java.sql.Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (rs.next())
{
    // print the values for the current row.
    int i = rs.getInt("a");
    String s = rs.getString("b");
    byte b[] = rs.getBytes("c");
    System.out.println("ROW = " + i + " " + s + " " + b[0]);
}
```

# Lecture des résultats de l'interrogation (3)

- La méthode `getMetaData()` permet d'obtenir les méta-données d'un `ResultSet`.
- Elle renvoie des `ResultSetMetaData`.
- On peut connaître :
  - Le nombre de colonne : `getColumnCount()`
  - Le nom d'une colonne : `columnName(int col)`
  - Le type d'une colonne : `getColumnType(int col)`
  - ...



# Exemple d'interrogation

```
public class TestJDBC {  
    public static void main(String[] args) throws Exception {  
        Class.forName("postgresql.Driver");  
        Connection conn = DriverManager.getConnection("jdbc:postgresql:mabase",  
  "dubois", "");  
  
        Statement stmt = conn.createStatement();  
        ResultSet rs = stmt.executeQuery("SELECT * from employe");  
        while (rs.next()) {  
            String nom = rs.getString("nom");  
            String prenom = rs.getString("prenom");  
            String email = rs.getString("email");  
        }  
        rs.close();  
        stmt.close();  
    }  
}
```

# Exécution d'une requête de mise à jour

- Execution de la requête :

```
String myQuery = "INSERT INTO employe " +  
                 "(prenom, nom, email) VALUES " +  
                 "('Michel', 'DUBOIS', 'duboism@iu-  
vannes.fr')";
```

```
int nb_line= stmt.executeUpdate(myQuery);
```

nb\_line : nombres de lignes mises à jour dans la base

# Mise à jour et transaction

- Par défaut, la connexion valide chaque mise à jour après un ordre `executeUpdate`. Chacun de ces ordres sont donc considérés comme élémentaires.
- Exemples d'ordre non élémentaire : mise à jour de données comptables (principe de la comptabilité en partie double): débit/crédit, commande/sortie de stock, etc.

Connection

```
con=DriverManager.getConnection(url,user,passwd);
con.setAutoCommit(false);
Statement stmt=con.createStatement();
stmt.executeUpdate("insert into ...");
...
stmt.executeUpdate("update ...");
if (ok==true) {con.commit();} else {con.rollback();}
```

# Transaction et concurrence

- On peut fixer le niveau d'isolation d'une transaction.

```
Connection
```

```
con=DriverManager.getConnection(url,user,passwd);
```

```
//niveaux d'isolation
```

```
con.setTransactionIsolation(TRANSACTION_READ_COMMITTED);
```

```
con.setTransactionIsolation(TRANSACTION_SERIALIZABLE);
```

# Gestion des Exceptions

- Rien de neuf sauf que la probabilité des `SQLException` est grande tant l'interface avec un SGBD est un processus délicat.
- De plus la quasi totalité des méthodes prédéfinies lancent une exception de type `SQLException`. Il convient donc de gérer l'occurrence de cet événement grâce à l'instruction `catch`.
- Une `SQLException` peut en cacher une autre. La méthode `getNextException()` permet de traiter la suivante (`getMessage()`).
- À noter que `Class.forName` lance une `ClassNotFoundException`.
- À noter que l'ouverture d'une `Connection`, d'un `Statement` ou d'un `ResultSet` entraîne normalement leur fermeture. En cas de sortie précipité d'un bloc `try`, un bloc `finally` sera exécuté quoi qu'il arrive (`catch`, `break`, `continue`, `return`).

# Les requêtes pré-compilées

Comme elles sont pré-compilées, leur exécution est plus rapide. La pré-compilation est donc intéressante pour les requêtes fréquentes.

Des paramètres IN permettent de modifier l'exécution dynamique de la requête pré-compilée.

Une conversion de type des paramètres s'opère en fonction d'une table de correspondance des types Java et SQL.

```
PreparedStatement ps=con.createStatement("update t1 set  
cat=? where age=?");  
ps.setString(1,"Ado");  
for (int i=14; i<19;i++) {  
    ps.setInt(2,i);  
    int nbTules=ps.executeUpdate();  
}
```

Michel Dubois 2011-2012

# Les types SQL 92 & java

| <u>JDBC type (java.sql.Types)</u> | <u>Java type</u>     |
|-----------------------------------|----------------------|
| CHAR, VARCHAR, LONGVARCHAR        | String               |
| NUMERIC, DECIMAL                  | java.math.BigDecimal |
| BIT                               | boolean              |
| TINYINT                           | byte                 |
| SMALLINT                          | short                |
| INTEGER                           | int                  |
| BIGINT                            | long                 |
| REAL                              | float                |
| FLOAT, DOUBLE                     | double               |
| BINARY, VARBINARY, LONGVARBINARY  | byte[]               |
| DATE                              | java.sql.Date        |
| TIME                              | java.sql.Time        |
| TIMESTAMP                         | java.sql.Timestamp   |

# Les procédures stockées

Les procédures stockées peuvent avoir des paramètres OUT.

```
CREATE OR REPLACE PACKAGE GererFnuc AS
    FUNCTION AUTH_CLIENT (nom_client IN clients.nom%TYPE, motdepasse_client IN
        clients.motdepasse%TYPE)RETURN clients.id%TYPE;
END GererFnuc;
```

...

```
CallableStatement cstmt=con.prepareStatement("{? = call
        GererFnuc.AUTH_CLIENT(?,?)}");
cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
cstmt.setString(2,nom);
cstmt.setString(3,password);
cstmt.execute();
clientId =cstmt.getInt(1);
cstmt.close();
```



# Correspondance de type SQL-Java

| Type SQL | Type Java            | Type Java<br>retourné par getObject() | Méthode recommandée<br>au lieu de getObject() |
|----------|----------------------|---------------------------------------|-----------------------------------------------|
| NUMERIC  | java.Math.BigDecimal | java.Math.BigDecimal                  | java.Math.BigDecimal<br>getBigDecimal()       |
| DECIMAL  | java.Math.BigDecimal | java.Math.BigDecimal                  | java.Math.BigDecimal<br>getBigDecimal()       |
| BIT      | boolean              | Boolean                               | boolean getBoolean()                          |
| TINYINT  | byte                 | Integer                               | byte getByte()                                |
| SMALLINT | short                | Integer                               | short getShort()                              |
| INTEGER  | integer              | Integer                               | integer getInt()                              |
| BIGINT   | long                 | Long                                  | long getLong()                                |
| REAL     | float                | Float                                 | float getFloat()                              |
| FLOAT    | double               | Double                                | double getDouble()                            |
| DOUBLE   | double               | Double                                | double getDouble()                            |

**Doit être utilisé pour les valeurs  
monétaires !! (plutôt que Float)**

# Correspondance de type SQL-Java

| Type SQL                             | Type Java                                            | Type Java<br>retourné par getObject()                | Méthode recommandée<br>au lieu de getObject()                                                                    |
|--------------------------------------|------------------------------------------------------|------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| CHAR<br>VARCHAR<br>LONGVARCHAR       | String<br>String<br>String                           | String<br>String<br>String                           | String getString()<br>String getString()<br>InputStream<br>getAsciiStream()<br>InputStream<br>getUnicodeStream() |
| BINARY<br>VARBINARY<br>LONGVARBINARY | byte[]<br>byte[]<br>byte[]                           | byte[]<br>byte[]<br>byte[]                           | byte[] getBytes()<br>byte[] getBytes()<br>InputStream<br>getBinaryStream()                                       |
| DATE<br>TIME<br>TIMESTAMP            | java.sql.Date<br>java.sql.Time<br>java.sql.Timestamp | java.sql.Date<br>java.sql.Time<br>java.sql.Timestamp | java.sql.Date getDate()<br>java.sql.Time getTime()<br>java.sql.Timestamp<br>getTimestamp()                       |

2/6

# Types de données Blob et Clob

- BLOB : Binary Large Object
- CLOB : Character Large Object

- Les Blob et les Clob pallient aux limites des types SQL VARCHAR et VARBINARY (capacité limitée)
- Les Blobs et les Clobs ne sont pas chargés lors d'un SELECT
- Pour les récupérer :

```
Blob b = rs.getBlob(1);  
InputStream binstr = b.getBinaryStream();  
Clob c = rs.getClob(2);  
Reader charstr = c.getCharacterStream();
```

- Les Blobs et les Clobs peuvent également être récupérés par morceaux

# JDBC 4.0 (JDK 1.6)

- Chargement automatique des drivers JDBC
- Nouvelles exceptions :
  - `SQLException`,
  - `SQLIntegrityConstraintViolationException` ...
- Support XML
- Utilisation des annotations Java 5

```
interface LoanAppDetailsQuery extends BaseQuery {  
    @Select("SELECT * FROM LoanDetails where LoanStatus  
    ='A'")  
    DataSet<LoanApplication> getAllActiveLoans();  
}
```

# SQL et Sécurité Web

- Attaque "SQL injection" à partir de formulaires Web

```
CREATE TABLE notes ( netudiant text, note float ) ;
```

- Formulaire traité JSP (idem PHP, ASP, ...)

```
String num = request.getParameter( "netudiant" ) ;  
String requete =
```

```
"SELECT * FROM notes WHERE netudiant='"+num  
ResultSet rs = st.executeQuery(requete);
```

- Saisie ' ; SELECT \* FROM notes WHERE ''=''  
requete =

```
SELECT * FROM notes WHERE  
netudiant=""; SELECT * FROM notes WHERE "="
```

- ! toutes les notes !!

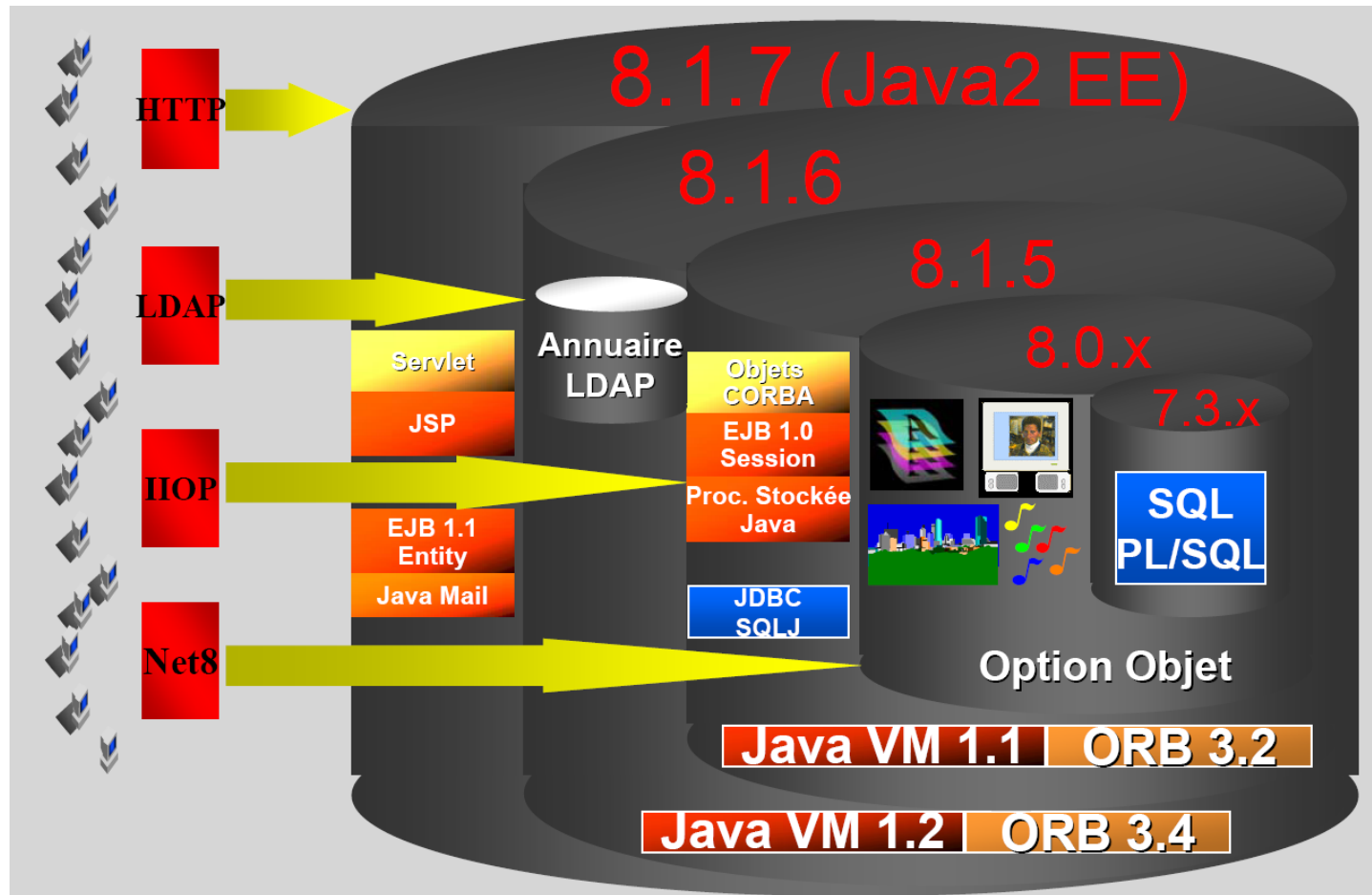


# Plan

- Mise en œuvre avec les pilotes JDBC
- L ' API JDBC et son utilisation
  - Basique et avancée
  - Pilote interne Oracle dans JVM
- Aurora
  - Objet-relationnel et JDBC
- ORM avec un exemple commenté : la Fnuc

Michel Dubois 2011-2012

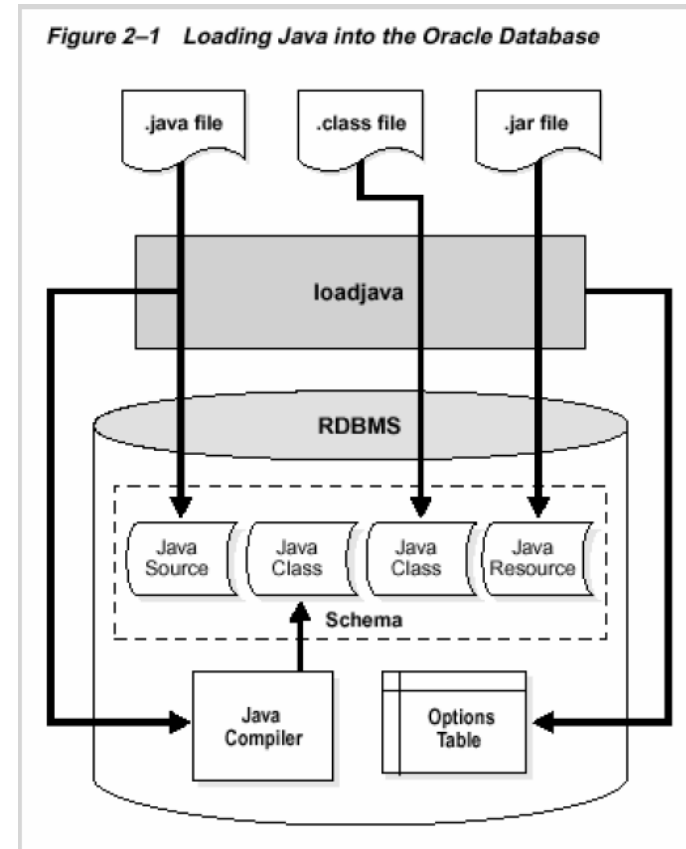
# Java dans Oracle 8i



# Les procédures stockées

Stocker du JAVA dans la base

- Source ou compilé
- Doit être publié
- Peut optionnellement être compilé pour améliorer les performances





# Etapas pour une procédure stockée

## Créer

```
public class Oscar {  
    // return a quotation from Oscar Wilde  
    public static String quote() {  
        return "I can resist everything except temptation.";  
    }  
}
```

## Charger

```
> loadjava -user scott/tiger Oscar.class
```

## Publier

```
SQL> connect scott/tiger  
SQL> CREATE FUNCTION oscar_quote RETURN VARCHAR2  
2 AS LANGUAGE JAVA  
3 NAME 'Oscar.quote { } return java.lang.String';  
/
```

## Exécuter

```
SQL> CALL oscar quote{} INTO :theQuote;  
SQL> PRINT theQuote;  
THEQUOTE
```

# Oracle, servlets et JSP

- Avec la version 8.1.7, le noyau ORACLE intègre un moteur de servlets et un moteur de JSP. C'est l'aboutissement du support complet de JAVA dans la base. (8i : JVM J2EE 1.2)
- Avec l'introduction Oracle Application Server Container for J2EE (OC4J), la version 9i R2 de la base de données n'a plus de moteur servlets et de conteneur de JSP et de EJB.
- Cependant, la JVM d'Oracle continuera à être améliorée pour les procédures stockées en java, JNDI, JDBC. (9i R2 : JVM J2SE 1.3 , 10g : JVM J2SE 1.4.1, 11g : JVM J2SE 1.5.2)

Michel Dubois 2011-2012

# JVM Oracle versus JVM SUN

- La JVM Oracle nommée Aurora permet à toute méthode statique d'être un point d'entrée (pas uniquement Main()).
- Utilise un utilitaire nommé « resolver » pour trouver les classes (pas de CLASSPATH). Le schéma PUBLIC est d'abord scruté, puis celui de l'utilisateur par le resolver par défaut.
- Offre un pilote JDBC interne.
- On peut charger soit les fichiers compilés .class soit les fichiers sources .java mais pas les 2. Oracle garde la trace du type de fichier chargé.
- On peut aussi charger des fichiers supplémentaires ( .properties, .txt, .jar )
- Les classes Java sont enregistrées comme objet du schéma
- Tous les . dans le nom d'une classe Java sont remplacés par /  
( fnuc.db.OracleJavaStore --- fnuc/db/OracleJavaStore )
- La taille du nom d'une classe Java doit être < 4000. La taille du nom d'un objet du schéma < 30 (si non, Oracle génère un alias). Pour connaître le nom complet : `FUNCTION DBMS_JAVA.LONGNAME (shortname VARCHAR2) RETURN VARCHAR2`

# SECURITE ORACLE 8I et JAVA

- JVM Aurora basé sur java 1.2
- Sécurité modèle java 1.2 avec contrôle d'accès aux ressources et aux objets
- Les classes java sont chargées dans un schéma de la base.
- Security policy définit dans une table sécurisée appartenant à l'utilisateur SYS. Les vues DBA\_JAVA\_POLICY et USER\_JAVA\_POLICY dépendent de cette table.
- Mise à jour de la table policy à travers les procédures DBMS\_JAVA.
- Rôle JAVA\_ADMIN pour modifier cette table
- Installation de Jserver par `initijvm.sql`.

# Pilotes JDBC Oracle

- JDBC Thin Driver :
  - Plus lent mais 100% pure Java
  - Complètement portable sans addin spécifique par plateforme ou par vendeur
  - Sans fonctionnalités propres aux vendeurs.
- JDBC OCI (Oracle Call Interface) :
  - Accès au mode natif Oracle
  - Plus rapide, plus riche en fonctionnalités mais plus gros.
- Pilote interne JDBC du côté serveur :
  - Utilisé par les procédures stockées en java
  - Permet l'accès aux données locales
  - Finement optimisée et support de tous les types Oracle
  - Le plus rapide, le plus petit, le moins consommateur de ressource réseau.
  - La connexion peut être obtenue par :

```
Connection conn =  
new oracle.jdbc.driver.OracleDriver().defaultConnection();
```
  - La connexion n'a pas à être fermée. Par contre, il faut fermer obligatoirement les Statement et Resultset.

# Compilation et chargement dans la base avec loadjava et dropjava

- **javac**, puis **loadjava MyClass.class**
  - compilation de manière explicite avant le chargement dans la base de données
  - création d'un objet JAVA CLASS
  - La manière la plus facile pour débogger mais attention à la version JVM
- **loadjava -resolve MyClass.java**
  - compilation du code source avec la JVM Oracle avec utilisation du resolver par défaut
  - création d'objet JAVA SOURCE et JAVA CLASS
  - permet une recompilation automatique en cas d'invalidation.
- **loadjava -resolve "((\* SCOTT)(\* PUBLIC))" -verbose MyClass.java**
  - même chose que précédemment mais le résolveur scrute d'abord le schéma SCOTT, puis le schéma PUBLIC
  - affiche des informations supplémentaires.
- **loadjava MyClass.java**
  - la compilation est différée jusqu'à la première exécution
  - création d'un objet JAVA SOURCE

loadjava accepte les fichiers .class, .java, .jar, .properties, .zip, .ser

(on peut aussi utiliser create java source, mais loadjava est préférable)

Si on utilise loadjava, il faudra utiliser dropjava pour enlever les classes du schéma. Ces deux utilitaires utilisant des tables (CREATE\$JAVA\$LOB\$TABLE et JAVA\$CLASS\$MD5\$TABLE) dans le schéma de l'utilisateur.

# Compilation et chargement dans la base avec le DDL Oracle

- `CREATE OR REPLACE JAVA SOURCE NAMED "xxx" AS`  
`code source`  
`;`  
`/`
- `CREATE OR REPLACE JAVA SOURCE NAMED "xxx"`  
`RESOLVER (( *SCOTT) ( *TEST) ( *PUBLIC))`  
`AS`  
`code source`  
`;`  
`/`
- `CREATE OR REPLACE AND COMPILE JAVA SOURCE NAMED "xxx" AS`  
`code source`  
`;`  
`/`
- `CREATE OR REPLACE JAVA CLASS USING BFILE ('c:\temp', 'xxx.class')`  
`;`  
`/`
- `ALTER JAVA SOURCE "xxx" COMPILE;`
- `SHOW ERRORS JAVA SOURCE "xxx";`
- `DROP JAVA SOURCE "xxx";`
- `DROP JAVA CLASS "xxx";` Michel Dubois 2011-2012

# System.out.println() et AURORA

- Rediriger la sortie vers le tampon DBMS\_OUTPUT :
  - SET serveroutput on size 100000
  - EXEC dbms\_java.set\_output(2000);



# La classe TestJDBC

```
import java.sql.*;

public class TestJDBC
{
    public static final String DB_URL = "jdbc:default:connection: ";
    public static final String SQL_TEXT = "select * from dept where deptno = ?";

    public static String getDeptName( int deptno ) throws SQLException
    {
        Connection conn = DriverManager.getConnection( DB_URL );
        // or ... = DriverManager( DB_URL, "scott", "tiger" );
        PreparedStatement pStmt = conn.prepareStatement(SQL_TEXT);
        pStmt.setInt( 1, deptno );
        ResultSet rSet = pStmt.executeQuery();
        rSet.next();
        return rSet.getString("dname");
    }
}
```

# Manipulations de TestJDBC

Après avoir sauvegardé cette classe dans le fichier TestJDBC.java

```
>loadjava -resolve -user scott/tiger@iup TestJDBC.java
> sqlplus scott/tiger@iup
```

```
SQL> SELECT SUBSTR(OBJECT_NAME,1,32) AS OBJECT_NAME, OBJECT_TYPE, STATUS FROM USER_OBJECTS WHERE OBJECT_TYPE LIKE 'JAVA%';
```

| OBJECT_NAME | OBJECT_TYPE | STATUS |
|-------------|-------------|--------|
| TestJDBC    | JAVA CLASS  | VALID  |
| TestJDBC    | JAVA SOURCE | VALID  |

```
SQL> CREATE OR REPLACE FUNCTION TEST_JDBC( dept IN NUMBER )
2  RETURN VARCHAR2 AS
3  LANGUAGE JAVA NAME 'TestJDBC.getDeptName(int)
4  return java.lang.String'
5  ;
6  /
```

Fonction créée.

```
SQL> VARIABLE name VARCHAR2(20)
```

```
SQL> CALL test_jdbc(30) into :name;
```

Appel terminé.

```
SQL> print name
```

```
NAME
-----
SALES
```

```
SQL> SELECT test_jdbc(20) from dual;
```

```
TEST_JDBC(20)
-----
RESEARCH
```

```
SQL> EXIT
```

```
>dropjava -user scott/tiger@iup TestJDBC.java
```

# Publier des classes java via des spécifications d'appels

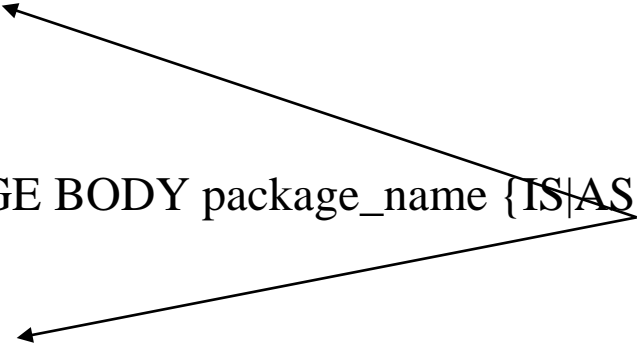
- Les méthodes ne sont pas publiées automatiquement lors du chargement car Oracle ignore les points d'entrée prévus par le programmeur.
- La publication des méthodes retournant une valeur se fera comme une fonction et les méthodes void comme des procédures.
- Le corps de la spécification contient la clause `JAVA`.
- Les incompatibilités sont détectées uniquement à l'exécution.

# Spécification d'appel autonome

```
CREATE [OR REPLACE]
{ PROCEDURE procedure_name [(param[,...])
  | FUNCTION function_name [(param[,...])
    RETURN sql_type }
[AUTHID {DEFINER|CURRENT_USER}]
[PARALLEL_ENABLE]
[DETERMINISTIC]
{IS | AS} LANGUAGE JAVA
NAME 'method_fullname (java_type_fullname[,...])
  [return java_type_fullname]';
where param is:
  parameter_name {IN|OUT|IN OUT} sql_type
```

# Spécification d'appel pour les procédures et les fonctions empaquetés

```
CREATE [OR REPLACE] PACKAGE package_name
    [AUTHID {DEFINER|CURRENT_USER}] {IS | AS}
    [type_definition]...
    [cursor_spec]...
    [item_declaration]...
    [{subprogram_spec|call_spec}]...
END [package_name];
[CREATE [OR REPLACE] PACKAGE BODY package_name {IS|AS}
    [type_definition]...
    [cursor_spec]...
    [item_declaration]...
    [{subprogram_spec|call_spec}]...
[BEGIN
    sequence_of_statements]
END [package_name]; ]
```



Dans un de ces  
2 emplacements  
mais pas dans  
les deux !

# Une classe pour les séquences

```
import java.sql.*;

public class CustomerInsertTrigger {

    public static void OnInsert(Connection conn, Long[] newid)
    {
        Long NextSequenceValue;

        NextSequenceValue = new Long(0);

        try{
            Statement stmt = conn.createStatement ();

            // Select the next id from the sequence
            ResultSet rset = stmt.executeQuery ("select
                customer_seq.nextval from dual");

            // Iterate through the results
            while (rset.next ())
                NextSequenceValue = new Long(rset.getLong(1));

            newid[0] = NextSequenceValue;

            // Close the ResultSet
            rset.close();
            rset = null;

            // Close the Statement
            stmt.close();
            stmt = null;
        } catch(SQLException e) {}
    }
}
```

# Déclencheurs en java

```
CREATE OR REPLACE PROCEDURE
CUSTOMER_INSERT( NEWID IN OUT NUMBER )
AS LANGUAGE JAVA NAME
    'CustomerInsertTrigger.OnInsert(
java.sql.Connection,
java.lang.Long[])';
/

CREATE OR REPLACE TRIGGER
    CUSTOMER_INSERT_TRIGGER BEFORE INSERT
    ON CUSTOMER FOR EACH ROW
    CUSTOMER_INSERT( new.CUSTOMER_ID );
/
```

# La politique déclarative de sécurité dans AURORA

- Erreur due à un manque de droits :

```
ERROR at line 1: ORA-29532: Java call terminated by
uncaught Java exception:
java.security.AccessControlException: the Permission
(java.io.FilePermission
/local/oracle/dmw_test/dave_test.in read) has not
been granted by dbms_java.grant_permission to
SchemaProtectionDomain(SCOTT|PolicyTableProxy(SCOTT))
ORA-06512: at "SCOTT.TEST4", line 0 ORA-06512: at
line 2
```

- Permettre à un utilisateur d'affecter des droits :

```
CALL DBMS_JAVA.GRANT_POLICY_PERMISSION('SCOTT',
'SYS', 'java.io.FilePermission', '*');
```

- Affecter des droits :

```
CALL
DBMS_JAVA.GRANT_PERMISSION('SCOTT','java.io.FilePermi
ssion', '/local/oracle/dmw_test/dave_test.in',
'read,write' );
```

```
COMMIT;
```

Michel Dubois 2011-2012



# Java DB / Apache Derby

- Java SE 6 intègre une base de données : Java DB dans le sous répertoire db du répertoire d'installation d'un JDK. C'est en fait la base de données open source Apache Derby écrite entièrement en Java. C'est une base de données légère (2 Mb) qui propose cependant des fonctionnalités intéressantes (gestion des transactions et des accès concurrents, support des triggers et des procédures stockées, ...)
- La version 10.2 permet de rapidement écrire des applications qui utilisent une base de données et les fonctionnalités proposées par la version 4.0 de JDBC.

# Plan

- Mise en œuvre avec les pilotes JDBC
- L 'API JDBC et son utilisation
  - Basique et avancée
  - Pilote interne Oracle dans JVM Aurora
  - Objet-relationnel et JDBC
- ORM avec un exemple commenté : la Fnuc

# Le schéma simplifié de la base fnuc (pas de large objects)

clients(id:int, nom:string, motdepasse:string, CACumul:float)

commandes(id:int, datecom:date, #article:int, #client:int, quantite:int)

stocks(id:int ,#article:int, niveau:int, securite:int)

livres( id:int, titre:string , auteurs:string, resume\_url:string,  
couverture\_url:string, prix:float)

sujets( id:int , libelle:string)

motscles( id:int , libelle:string)

livres\_motscles(#book\_id:int, #keyword\_id:int)

livres\_sujets(#book\_id:int, #topic\_id:int);

# INTERFACE AVEC PLUSIEURS SGBD

Deux solutions :

- Solution simple mais limitée : un accès identique.
  - on utilise un même schéma logique pour les deux SGBD,
  - on utilise les instructions SQL 92 et pas de fonctions propres à un SGBD (ou la syntaxe ESCAPE si le pilote le permet),
  - on utilise un paramètre connexion.
- Un accès différencié
  - plusieurs classes d'accès aux données
  - pour ordonner ces classes, il vaut mieux utiliser un package
  - on peut séparer la logique applicative de l'accès des données.

# INTERFACE AVEC PLUSIEURS SGBD

- On souhaite un couplage faible entre la logique applicative et l'accès des données (pour pouvoir modifier l'accès de données à la suite au changement de SGBD).
- Pour cela, logique applicative ne manipule pas directement la classe d'accès aux données. Il y a deux packages correspondant à la logique(`log`) et à l'accès aux données (`store`). La logique interagit avec l'interface `store.Store`. Cette interface possède toutes les méthodes d'accès aux données nécessaires à la logique. Chaque classe fournissant un accès aux données devra implémenter cette interface.

# INTERFACE AVEC PLUSIEURS SGBD

- Pour accéder aux données, la logique invoque les méthodes de l'interface `store.Store` sur une instance d'une classe implémentant cette interface.
- On ne peut pas créer cette instance dans la logique par un `new` sur cette classe car le changement de classe d'accès aux données engendrerait des modifications dans le code de la logique.
- Une solution consiste à délocaliser la création de ces instances dans la classe `store.StoreFactory`. Ainsi, lors du changement de base de données il suffira simplement de modifier la méthode `StoreFactory.getInstance()` pour que la logique interagisse avec la nouvelle classe d'accès aux données.

# Exemple (1/2)

- Exemple d'interface Store :

```
package store;  
public interface Store {  
    public void open() throws Exception;  
    public void close() throws Exception;  
    public Topic[] getAllTopics() throws Exception;  
    ...}
```

- La classe OracleStore

```
package store;  
public class OracleStore implements Store{  
    ...}
```

- La classe PostgresqlStore

```
package store;  
public class OracleStore implements Store{  
    ...}
```

# Exemple (2/2)

Exemple de classe StoreFactory :

```
package store;  
  
class StoreFactory {  
    static Store getInstance() {  
        return new PostgresqlStore();  
    }  
}
```

Exemple d'utilisation dans la logique :

```
package loc;  
  
class Main {  
    static void main(String args[]) {  
        ...  
        Store store = StoreFactory.getInstance();  
        ... }  
}
```



# Plan

- Mise en oeuvre avec les pilotes JDBC
- L 'API JDBC et son utilisation
- Un exemple commenté : la Fnuc