

# Mapping objet/ relationnel

# Plan

- Motivations
- Mapping modélisation objet -> relationnel
  - Quelques rappels sur le diagramme de classes UML
  - Transformation d'un diagramme de classes
  - Le niveau physique
- Mapping applicatif : en TP !

# Motivations

- Modélisation et développement objet
  - Standard de fait
  - UML (Unified Modeling Language), depuis le début des années 90
  - Langages de programmation objet, expansion depuis le début des années 80:
    - C++ (C with Classes), années 80
    - Java (1995)
    - C#, Python PHP, Ruby, Perl, Smalltalk, ...



C++



# Comment gérer la persistance des objets ?

- Persistence = BD
- BD Objet
  - Approche NF2 (Non First Normal Form), années 90
  - Fonctionnalités orientées-objet: encapsulation, classes, héritage, surcharge



O2, Gemstone, Ontos, Objet Store

- BD Objet-relationnelle, milieu des années 90
  - Fonctionnalités SQL 3



Oracle

- BD relationnelle

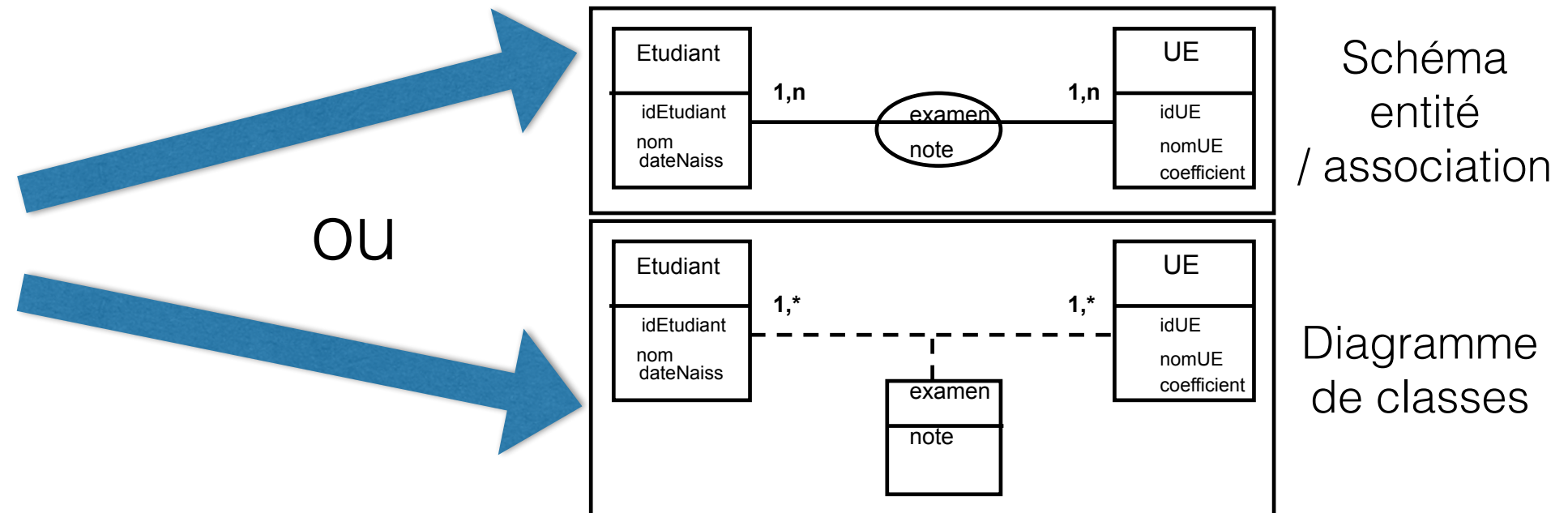
# Gestion de la persistance

- 2 niveaux
  - Modélisation
    - Passage d'une modélisation objet à un schéma relationnel
  - Applicatif
    - Utilisation d'un framework ORM (Object Relational Mapping)

# Modélisation d'une BD relationnelle



Monde Extérieur



## Schéma physique

**Mise en œuvre  
de la BD**

Langage  
SQL

**Etudiant** (idEtudiant, nom, dateNaiss)  
**Examen** (#idEtudiant, #idUE, note)  
**UE** (idUE, nomUE, coefficient)

## Schéma Logique ou Relationnel

# Le Diagramme de Classes UML (DC):

## Concepts de base

- Classe
  - Ensemble d'objets concrets ou abstraits de même nature
  - Une classe est décrite par ses attributs, méthodes et contraintes
  - Exemple: Etudiant, Employe, Produit,...
- Attribut
  - Propriété décrivant une classe
  - Valeur unique pour chaque classe
  - Exemple: nom, prenom, adresse, ...

# Le Diagramme de Classes UML (DC):

## Concepts de base

- Identifiant
  - Attribut particulier permettant de repérer une occurrence
  - Non obligatoire mais fortement recommandé dans notre contexte
  - Exemple: idClient, idEtudiant, ...
- Association
  - Permet de relier une classe à une ou plusieurs autres

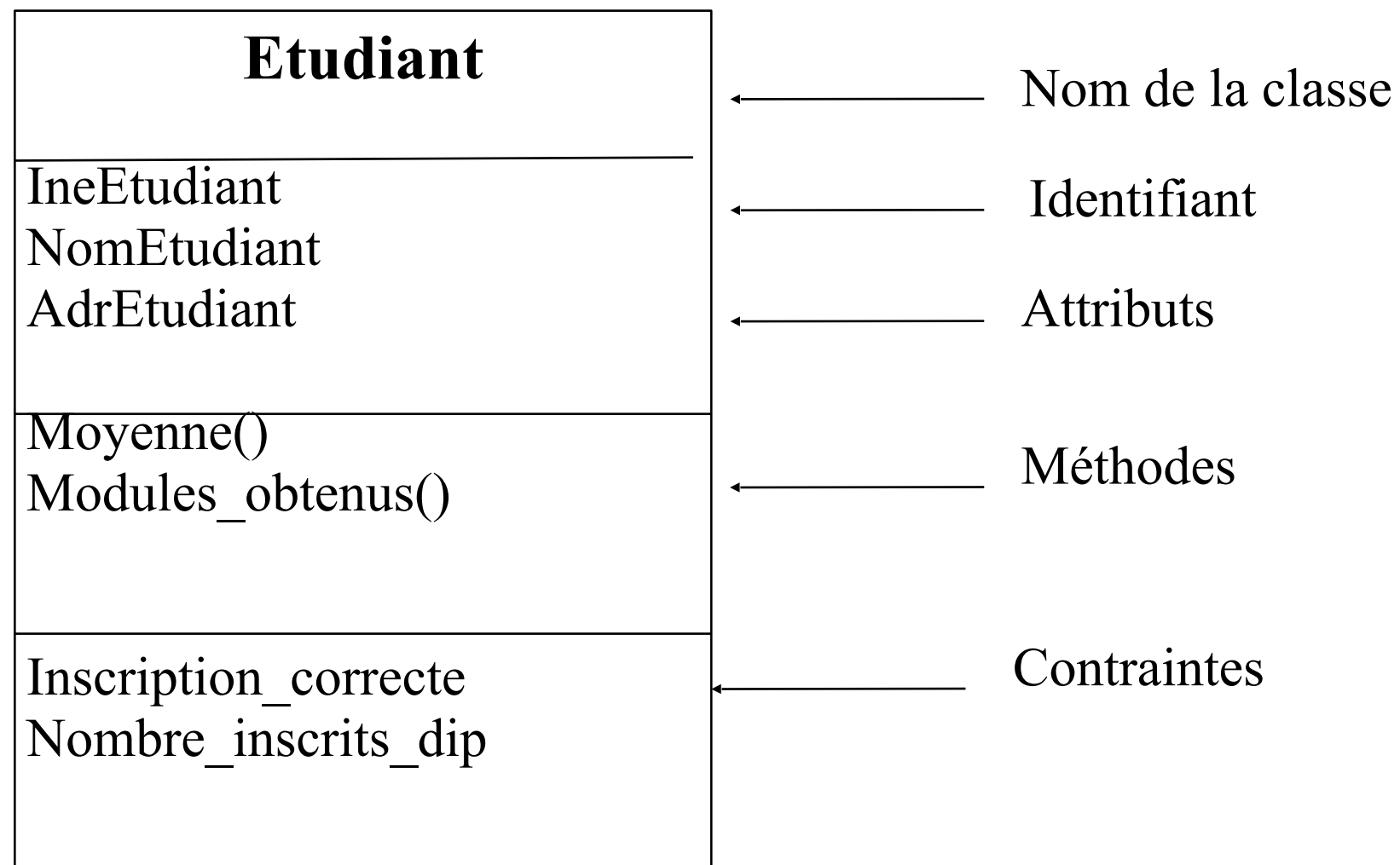


# Notion d'Association

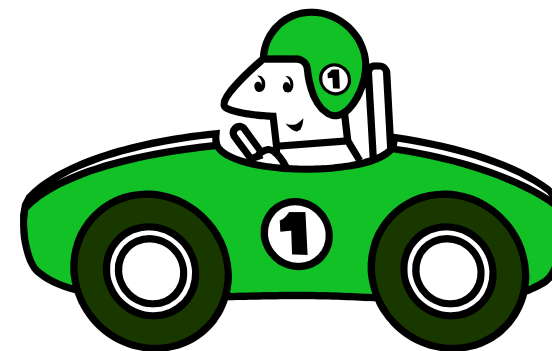
- Dimension d'une association
  - Nombre de classes liées
  - Binaire: 2; Ternaire:3; N-aire: n
- Nom d'une association
  - Souvent un verbe à l'infinitif: Appartenir, Fournir,...
- Multiplicité
  - Nombre minimum et maximum d'objets liés

Minimum Maximum	Optionnel	Obligatoire
Unicité	0..1	1..1 ou 1
Multiplicité	0..* ou *	1..*

# Représentation des classes

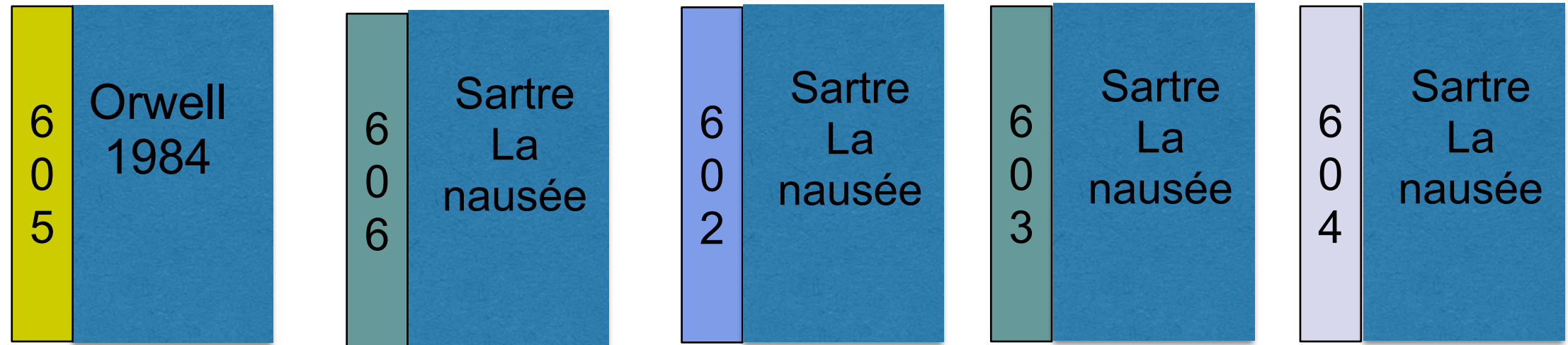


# Classes



- Engins roulants ?
- Véhicules à deux roues et véhicules à 4 roues ?

# Classes



Le littéraire voit deux livres d’auteurs classiques

OUVRAGE  
CLASSIQUE

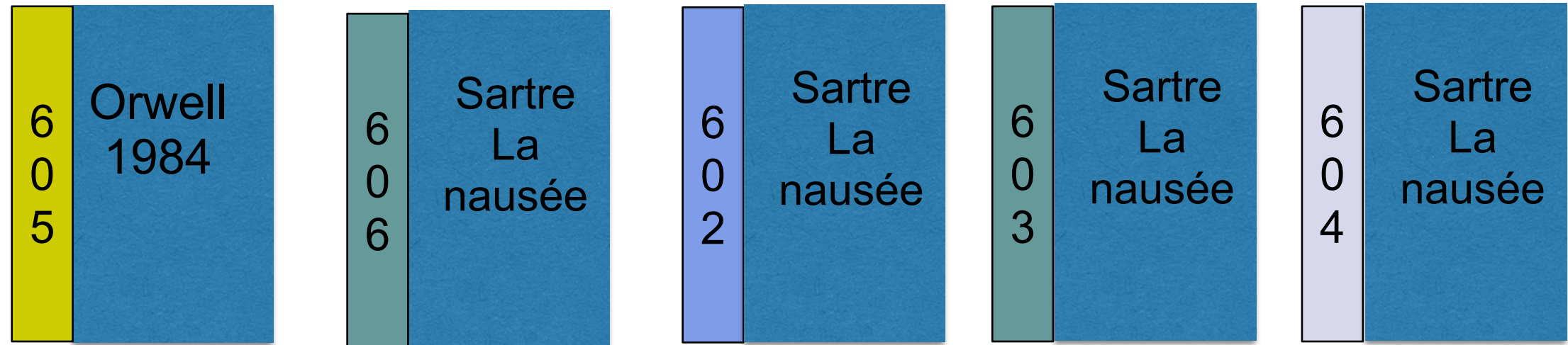
-Auteur  
-Titre

Le bibliothécaire voit cinq livres référencés dans ses rayons

LIVRE

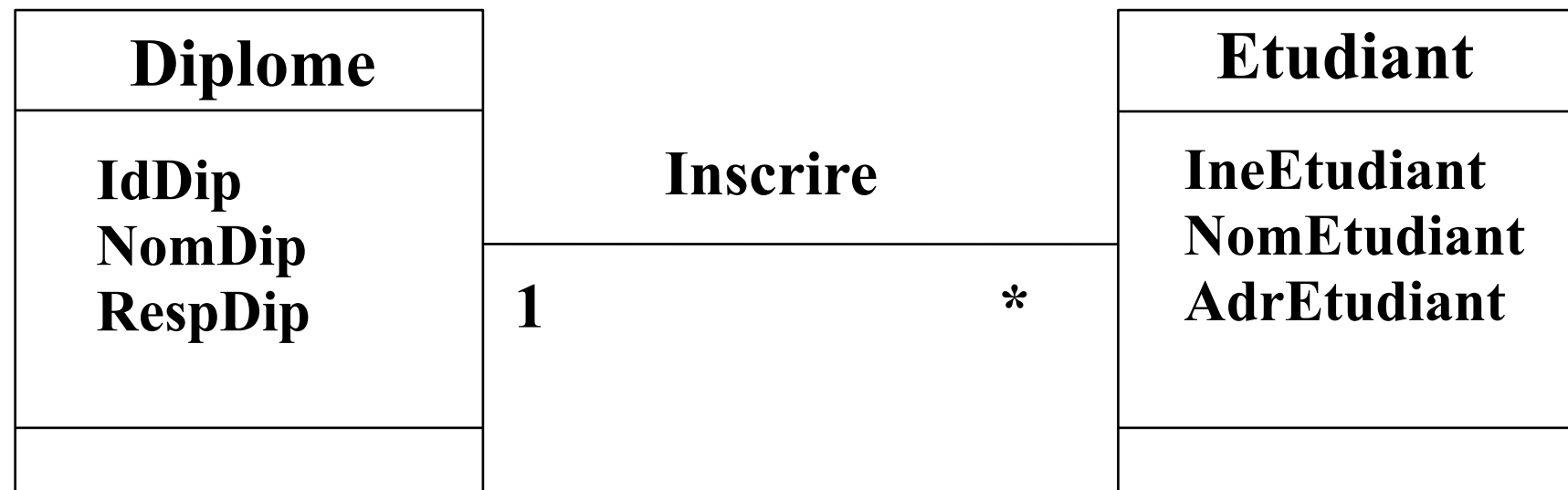
-Auteur  
-Titre  
-Côte

# Classes



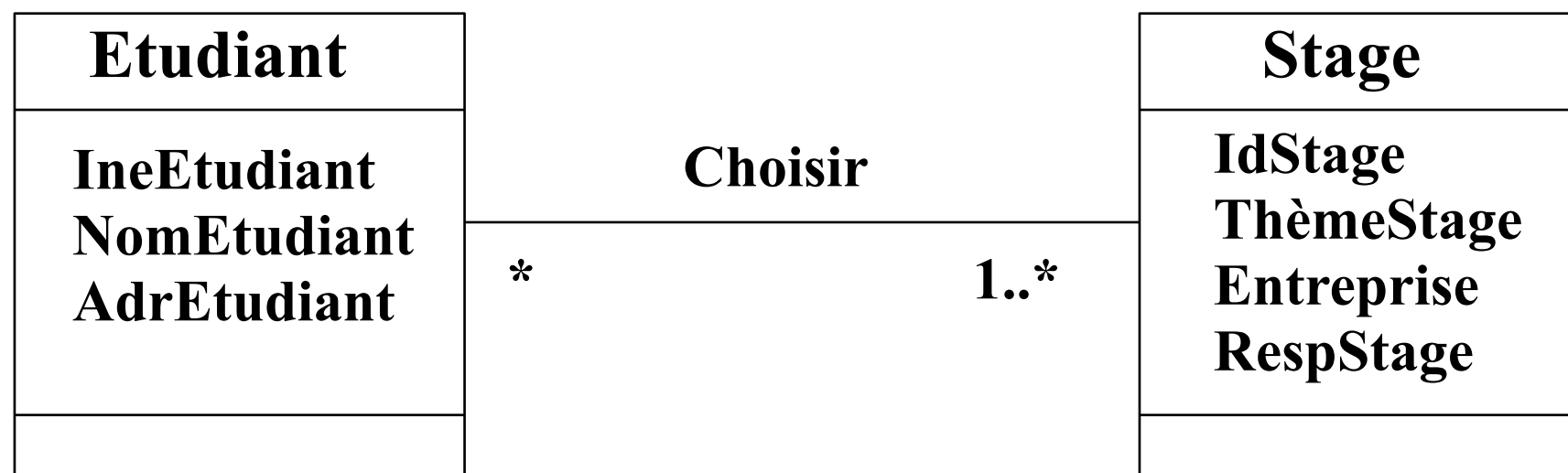
La définition de la classe est une décision prise par le concepteur, en fonction des **finalités** de l'entreprise

# Associations de type Mère-Fille (1-\*)



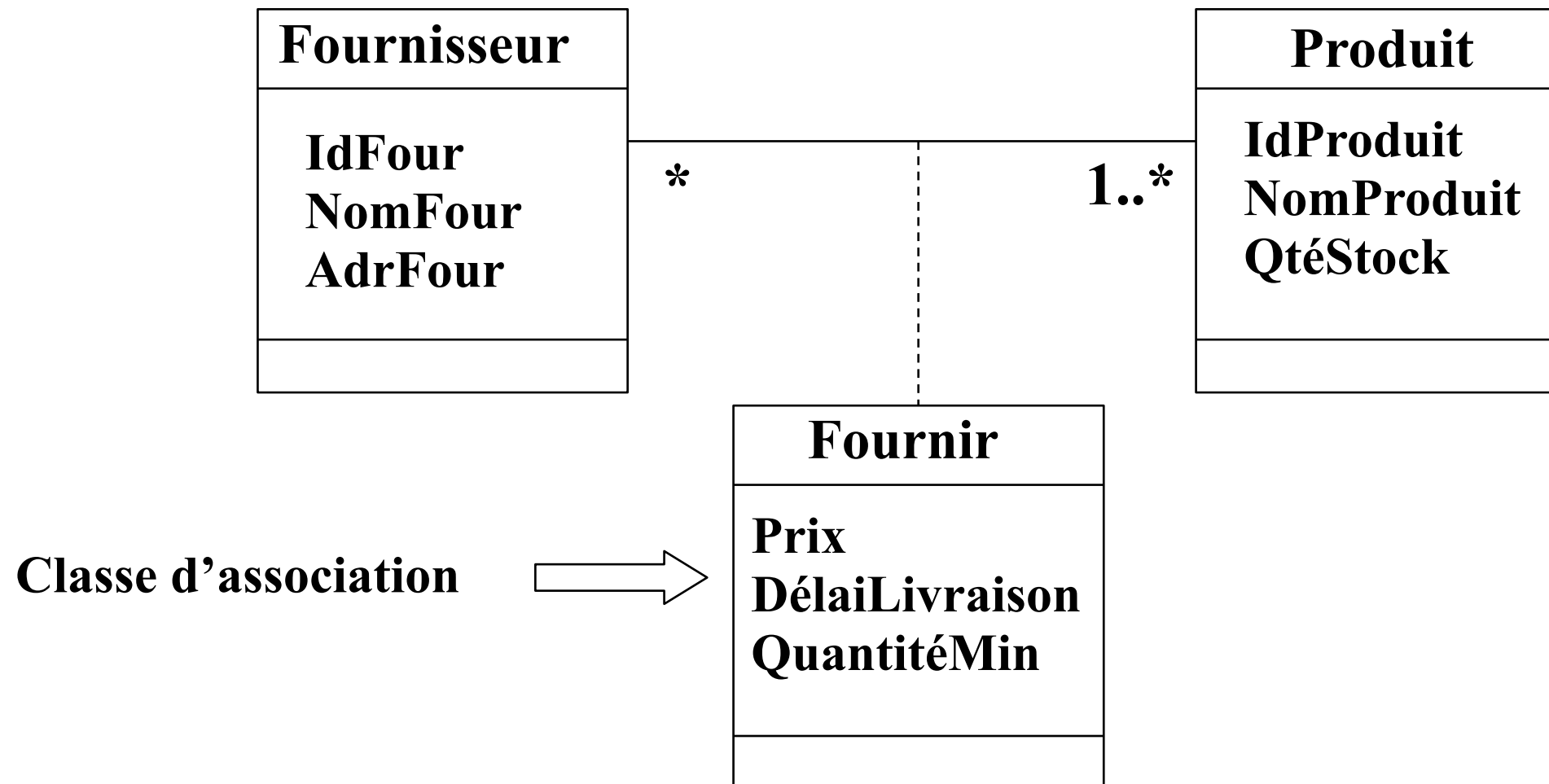
- **Un Diplôme** (classe mère) peut concerner aucun ou **plusieurs étudiants** (classe fille).
- **Un étudiant** doit être inscrit à, au moins et au plus, **un seul diplôme**

# Association de type multiple (\*-\*) sans attributs



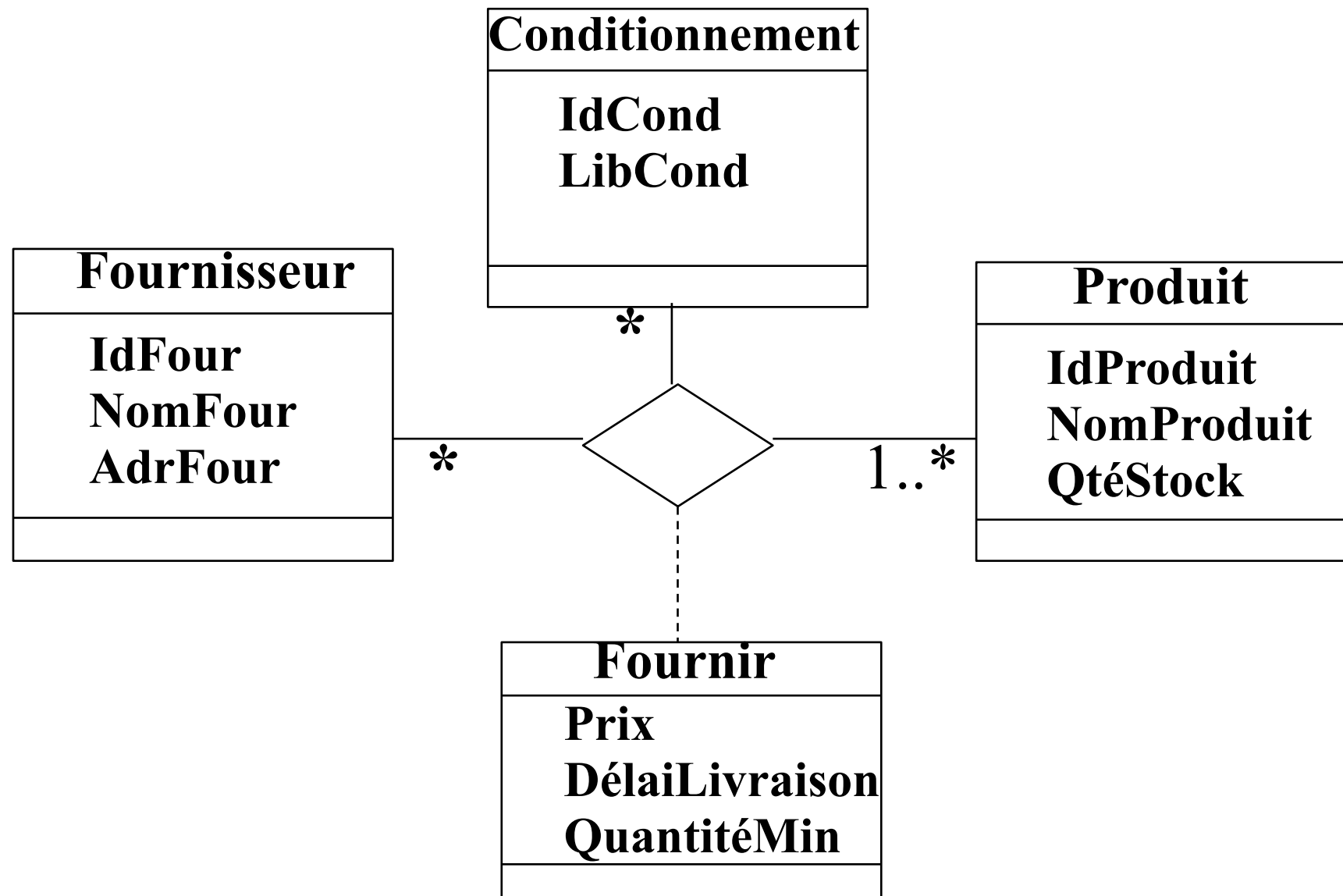
- **Un étudiant** doit choisir **au moins un stage** et peut en choisir **plusieurs**.
- **Un stage** peut être choisi par **aucun étudiant ou plusieurs**.

# Association de type multiple (\*-\*) avec attributs

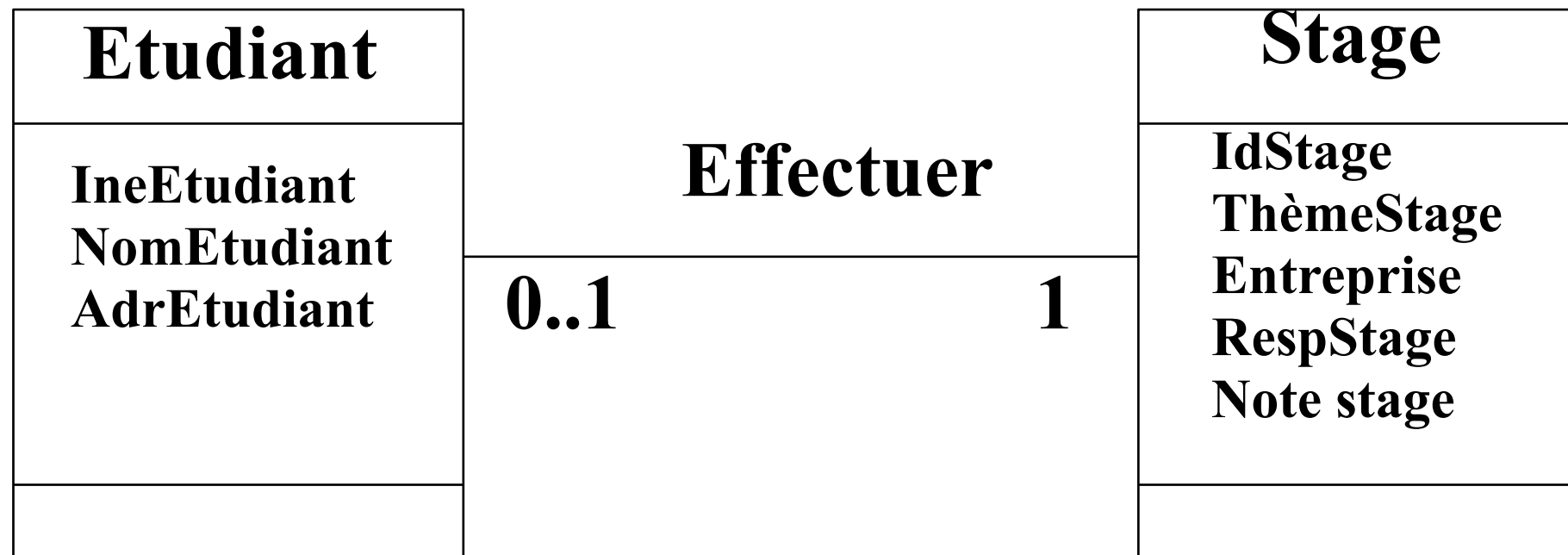




# Associations de type multiple (\*-\* ) N-aires (N>2)



# Associations de type symétrique (1-1)

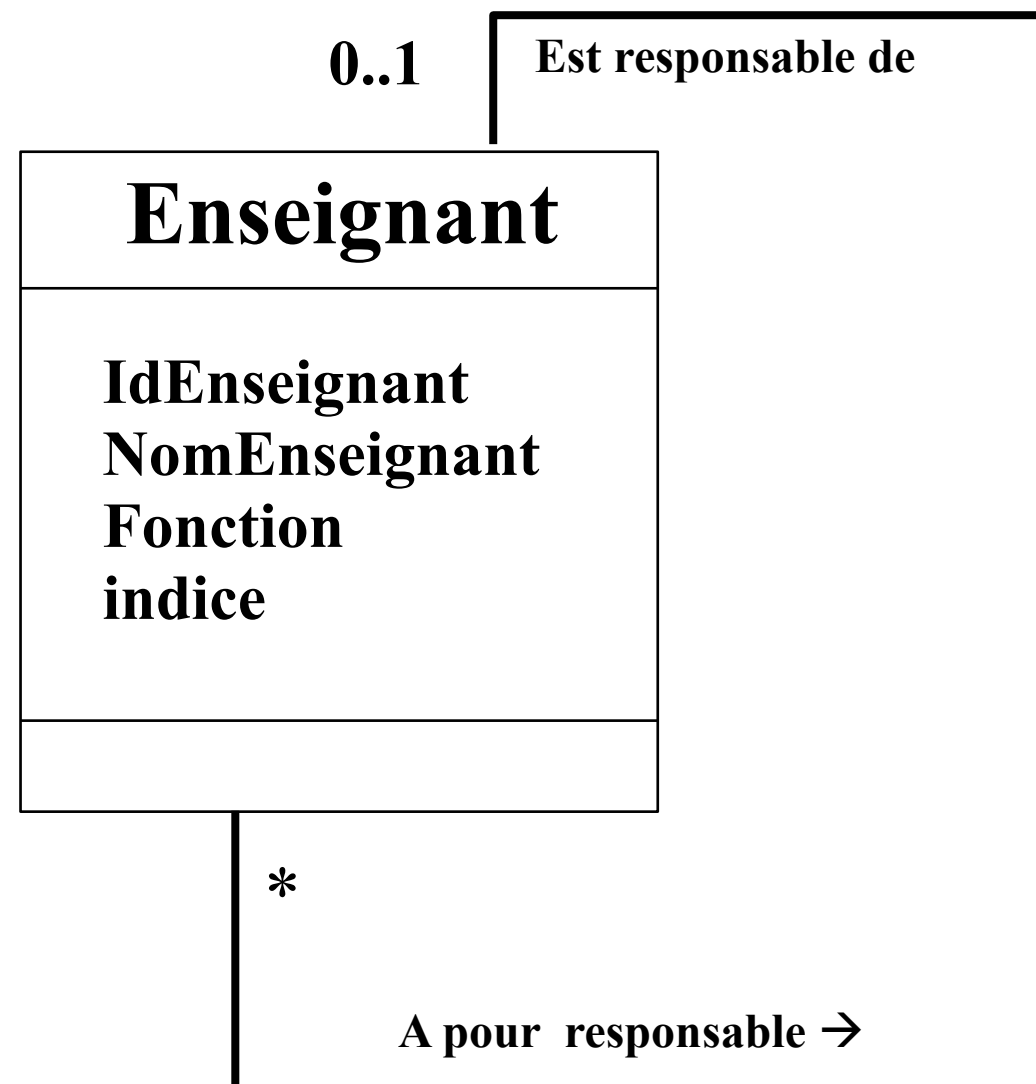


- **Un étudiant effectue un et un seul stage.**
- **Un stage peut être effectué par aucun ou un seul étudiant.**

Une association de type 1-1 est souvent le résultat d'un éclatement de classes.

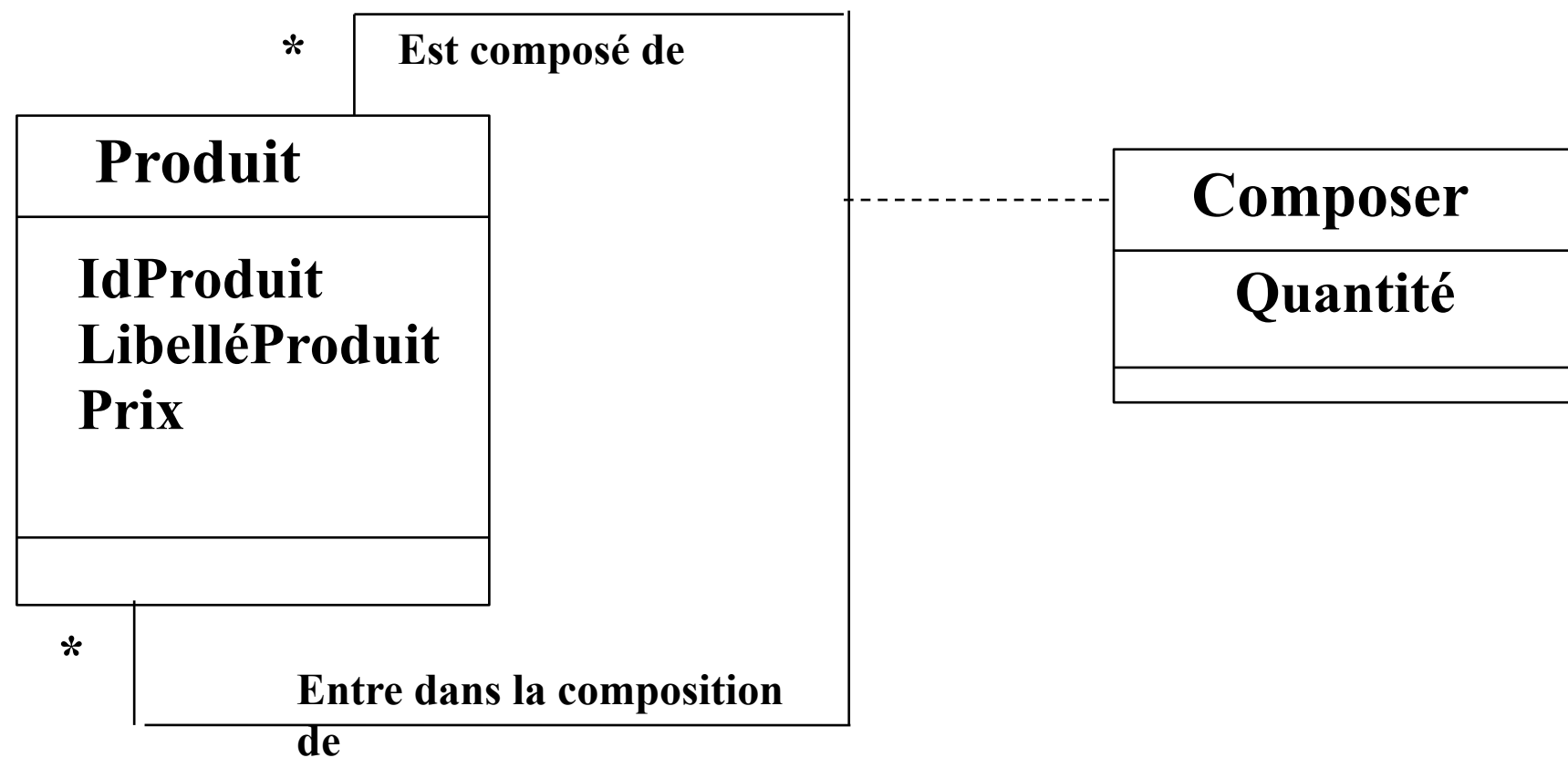
# Associations réflexives

-> 1-\* réflexif



# Associations réflexives

-> \*-\* réflexif



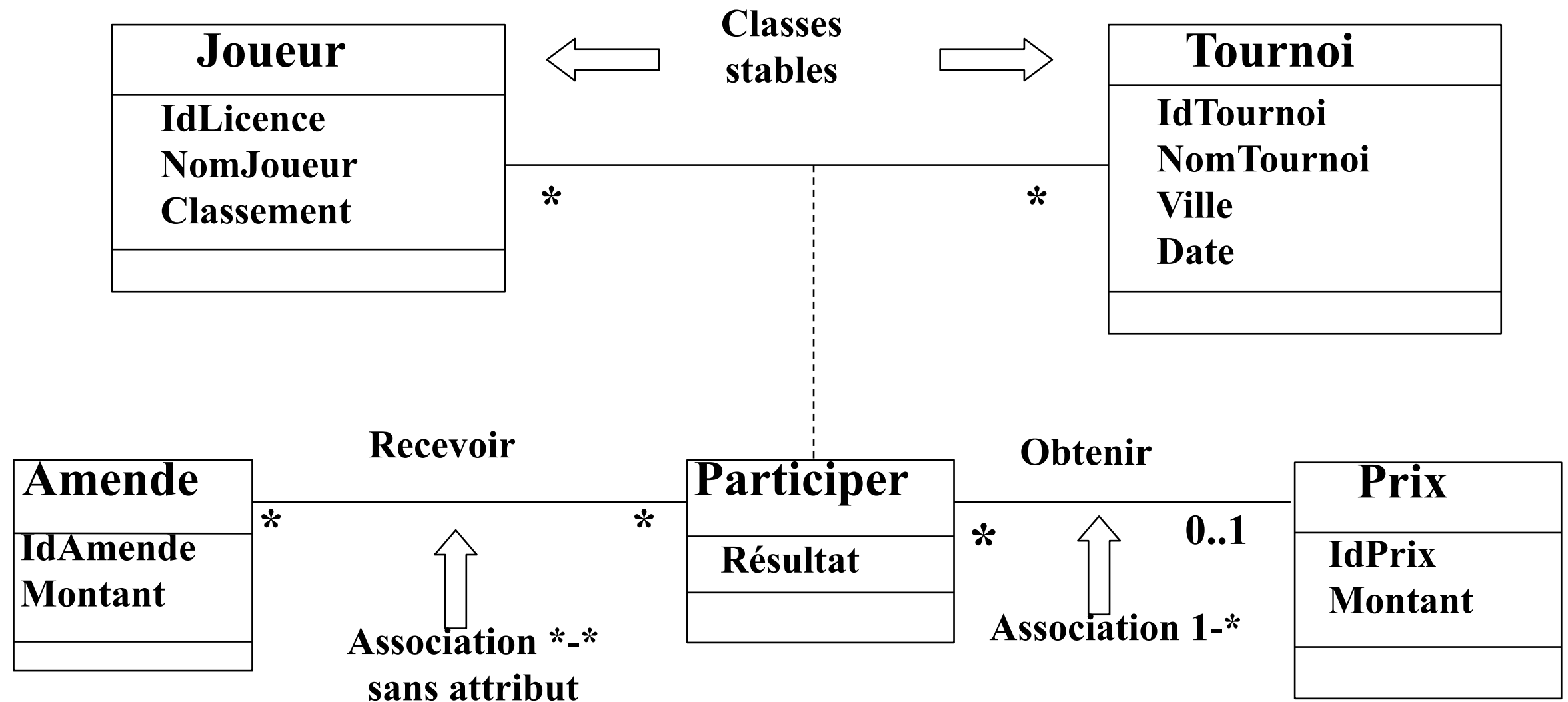
# Assemblage de classes

## Reprise d'une classe d'association

- Certaines associations N-aires peuvent être transformées en plusieurs associations binaires
- On associe d'abord les deux classes les plus stables: stabilité du schéma
- La classe d'association se transforme en classe normale pour la troisième classe associée

# Reprise d'une classe d'association

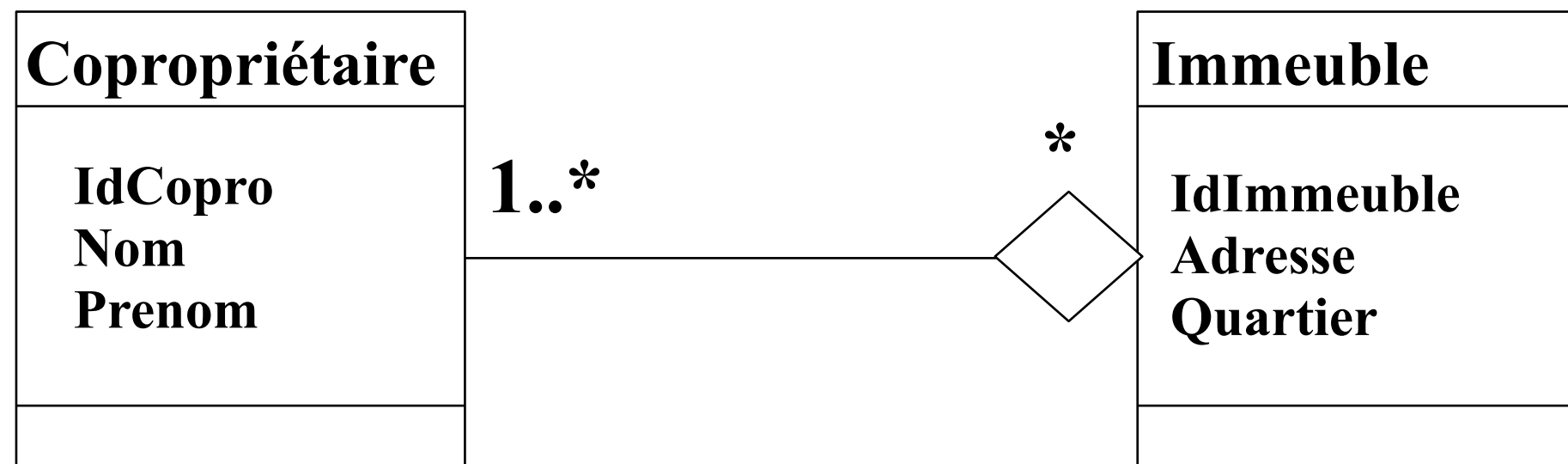
## Associations de type 1-\* et \*-\*



- La classe d'association 'Participer' est transformée en classe normale

# Agrégation

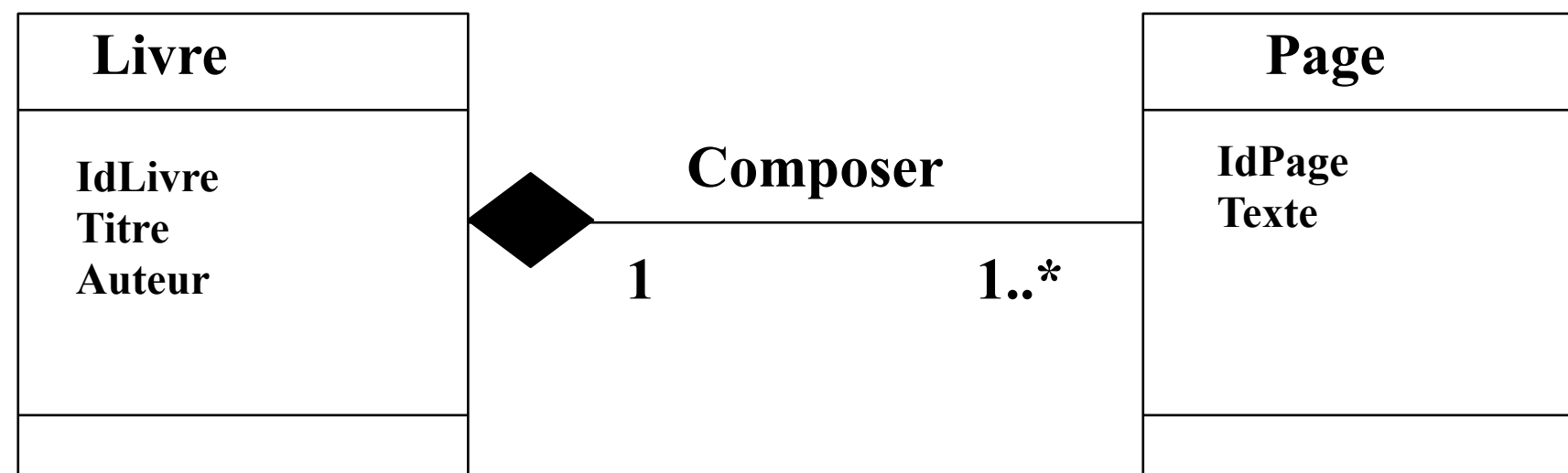
- Une association d'agrégation indique un principe de subordination entre l'agrégat et les agrégées
  - Elle indique une « possession »: l'agrégat peut contenir plusieurs objets d'un type



- L'existence de Copropriétaire n'a de sens que si l'immeuble existe déjà.
- La destruction d'un objet de type Immeuble n'entraînera pas forcément la destruction des copropriétaires, si ces derniers sont propriétaires dans d'autres immeubles

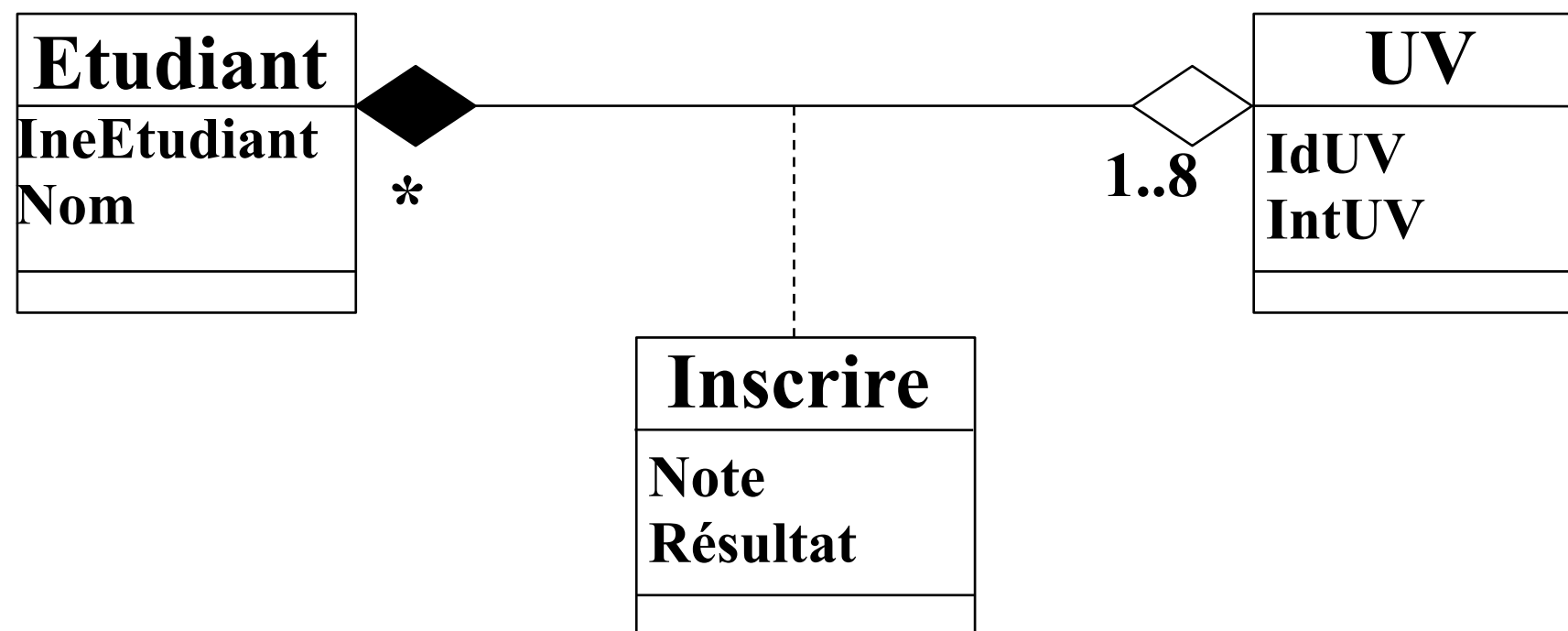
# Composition

- La composition est une agrégation à laquelle on impose des contraintes internes: un objet fait partie d'un seul composite
  - Les composants sont donc totalement dépendants du composite
  - Les cycles de vie des composants et de l'agrégat sont liés: si l'agrégat est détruit, ses composants le sont aussi





# Composition et agrégation sur classe d'association



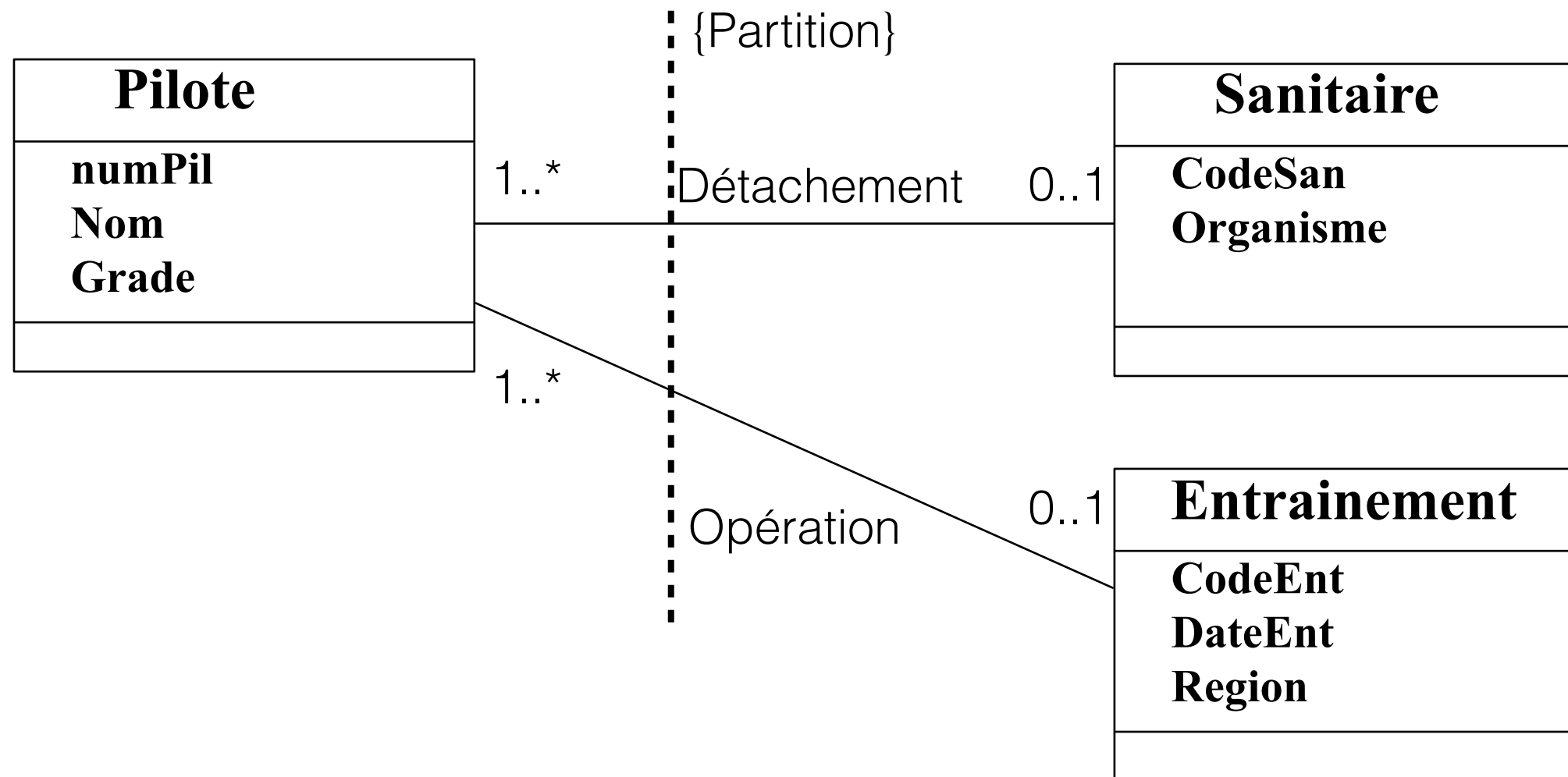
- La composition entre Etudiant et Inscrire indique qu'en cas de suppression d'un étudiant, on supprime toutes ces inscriptions
- L'agrégation traduit qu'on ne peut pas supprimer une UV ayant au moins un inscrit
  - On ne peut pas supprimer la classe mère tant qu'il reste des classes filles.

# Expression de contraintes

- But: améliorer la sémantique du DC
- Types de contraintes:
  - Contrainte de partition
  - Contrainte d'exclusion
  - Contrainte de totalité
  - Contrainte de simultanéité
  - Contrainte d'inclusion

# Expression de contraintes

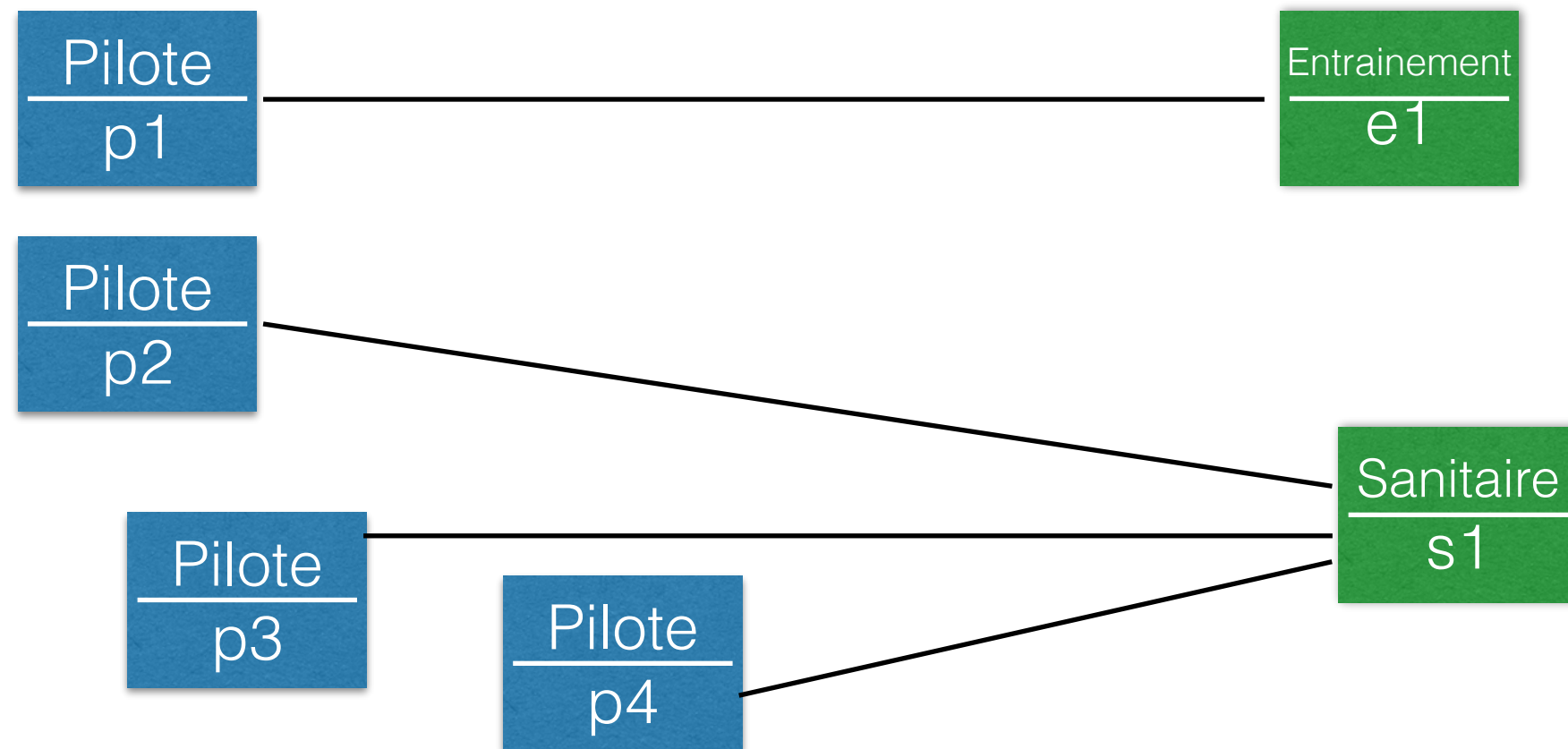
## Contrainte de partition



- Tous les objets d'une classe participent à l'un des deux associations, mais pas au 2, ni à aucune des deux

# Expression de contraintes

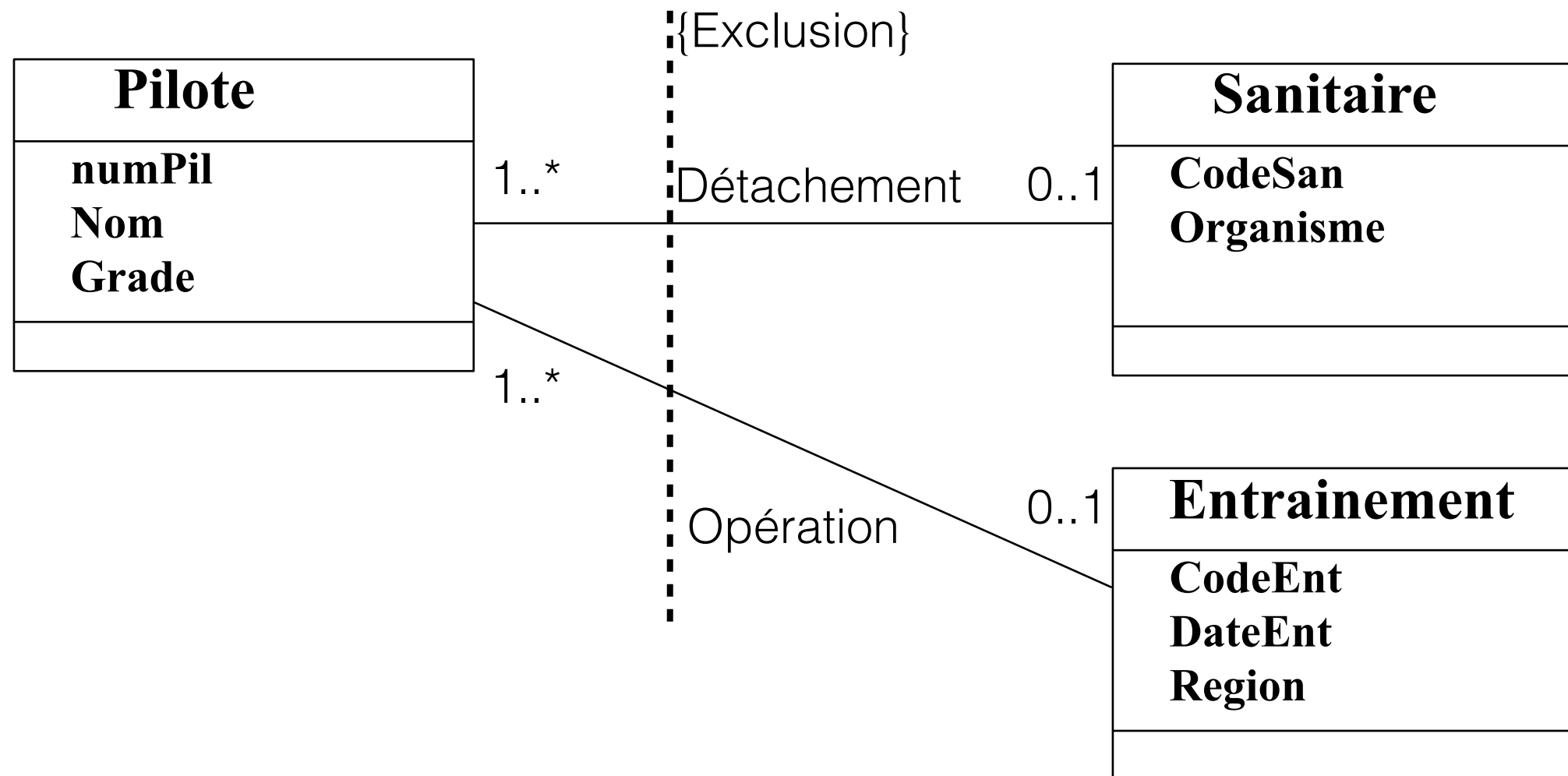
## Contrainte de partition - DO



- L'union des pilotes en mission sanitaire et en opération d'entraînement donne la totalité du contingent des pilotes.
- Aucun n'est au repos ni ne mène de front des missions des 2 types.

# Expression de contraintes

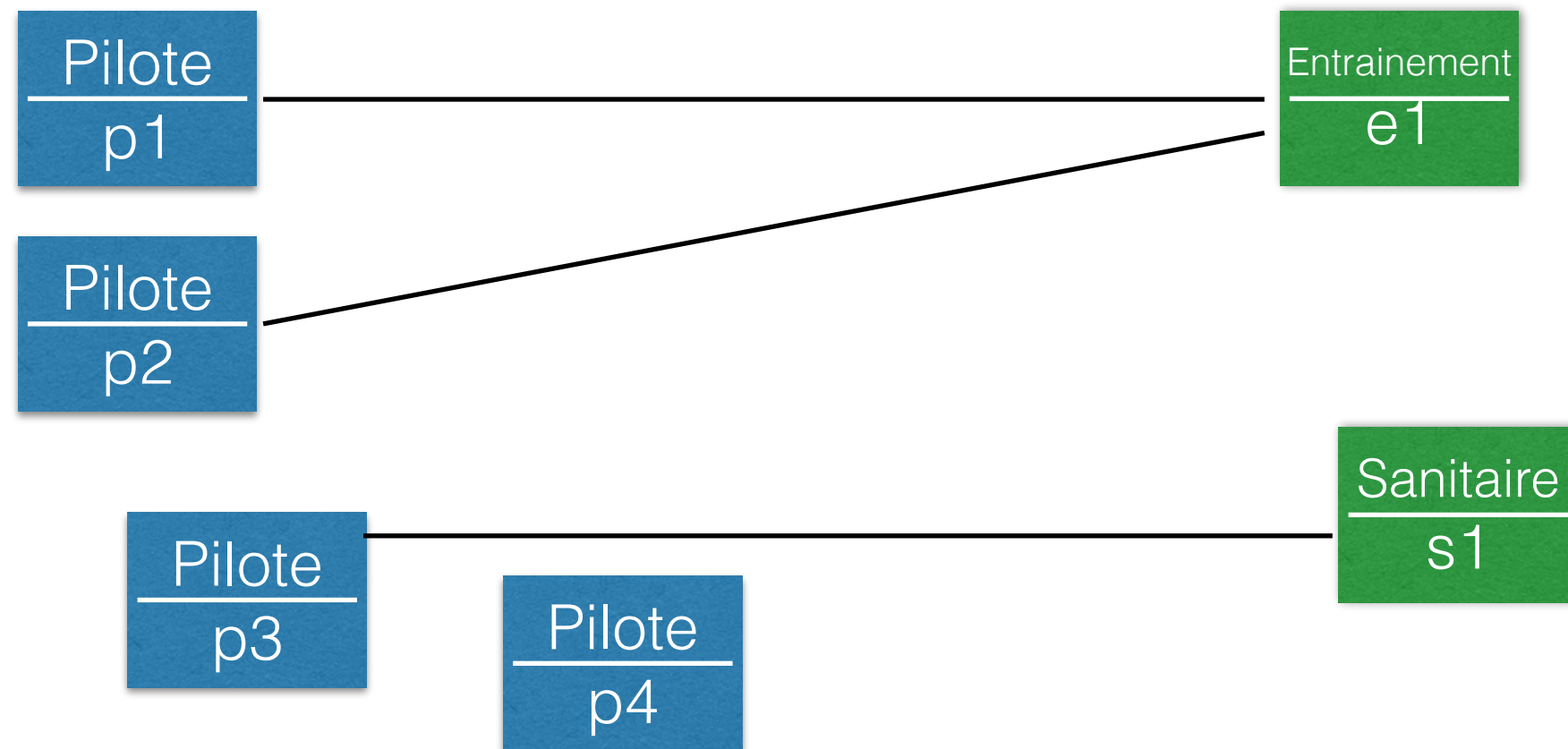
## Contrainte d'exclusion



- Tous les objets d'une classe peuvent participer à l'une des deux associations mais pas aux 2 à la fois

# Expression de contraintes

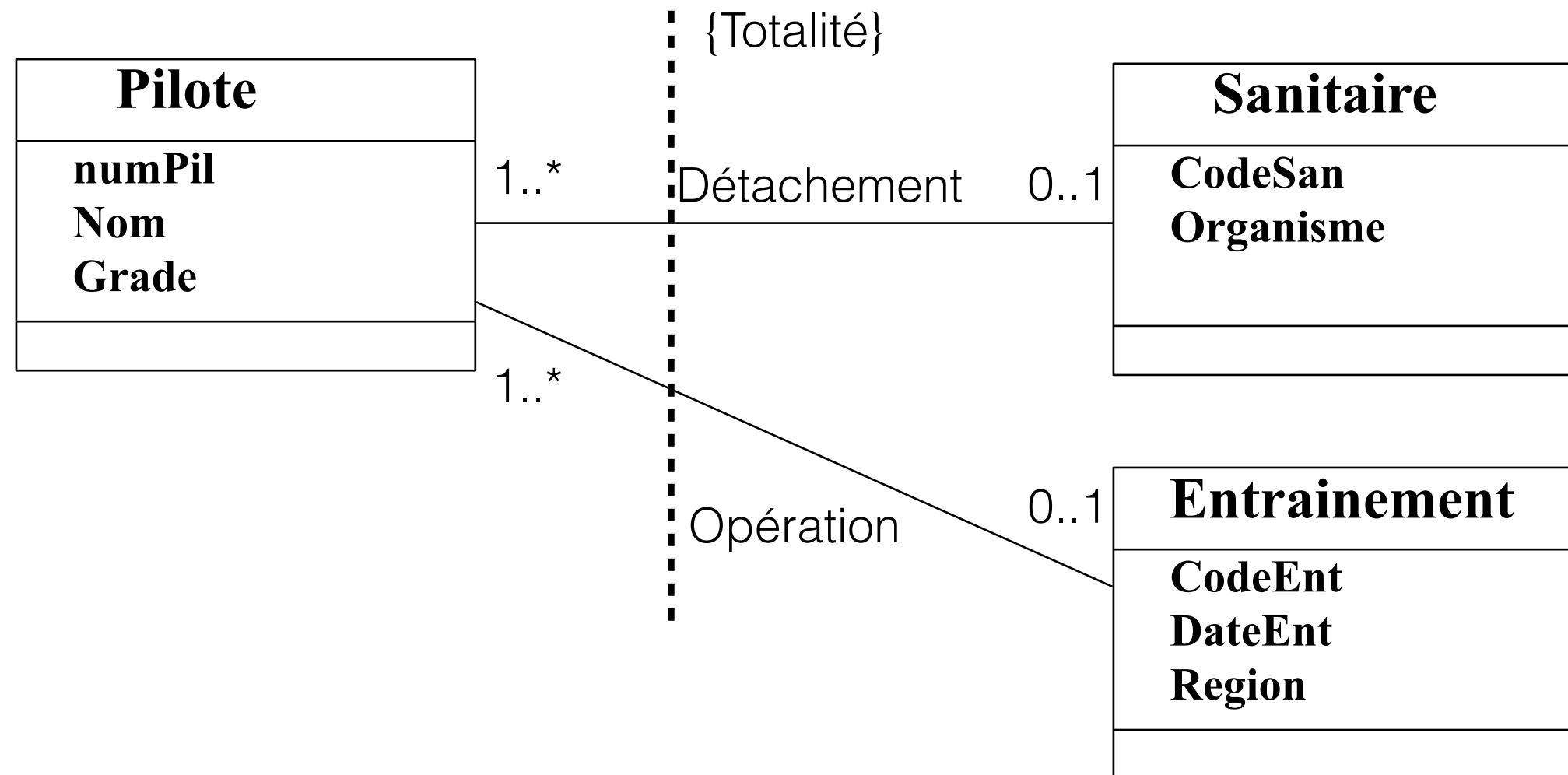
## Contrainte d'exclusion - DO



- Un pilote peut être au repos (il n'est affecté à aucune mission).
- S'il est affecté à un exercice d'entraînement, alors il ne peut pas être affecté à une mission sanitaire et réciproquement.

# Expression de contraintes

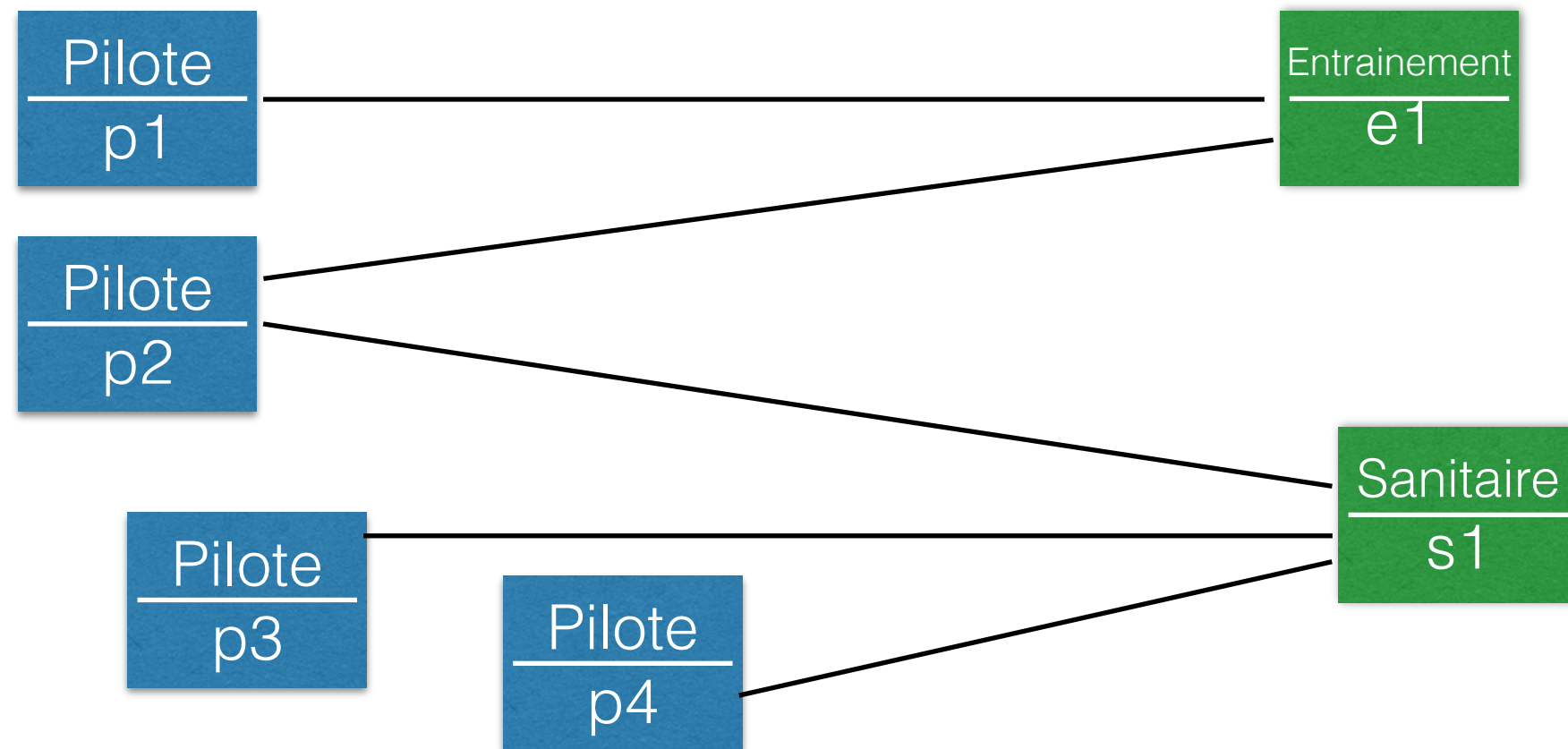
## Contrainte de totalité



- Tous les objets d'une classe participent au moins à une association

# Expression de contraintes

## Contrainte de totalité - DO

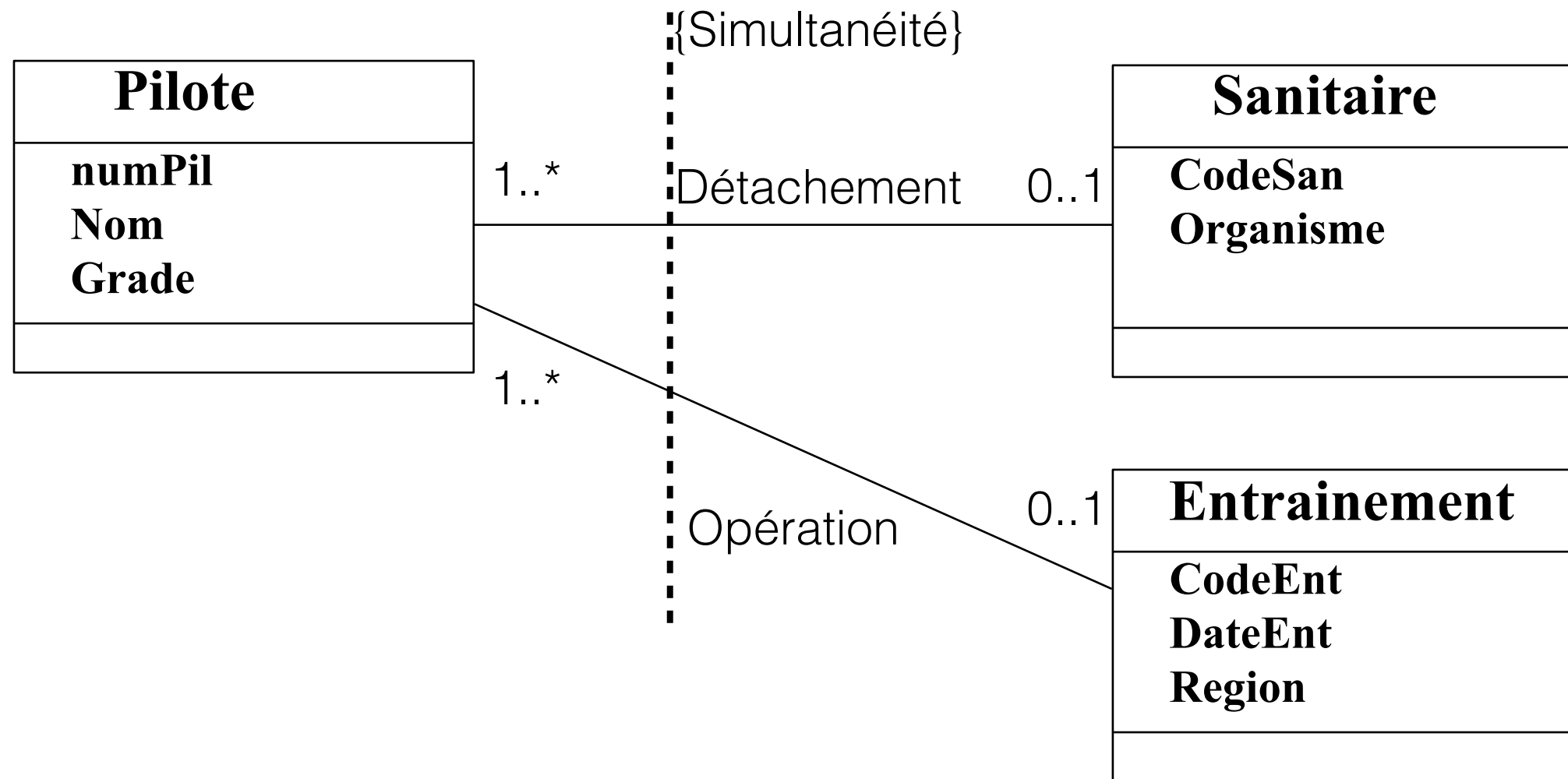


- Un pilote peut être affecté à la fois à une mission sanitaire et à une exercice d'entraînement et tous les pilotes participent à au moins une mission.
- Les pilotes forment donc la totalité du contingent.



# Expression de contraintes

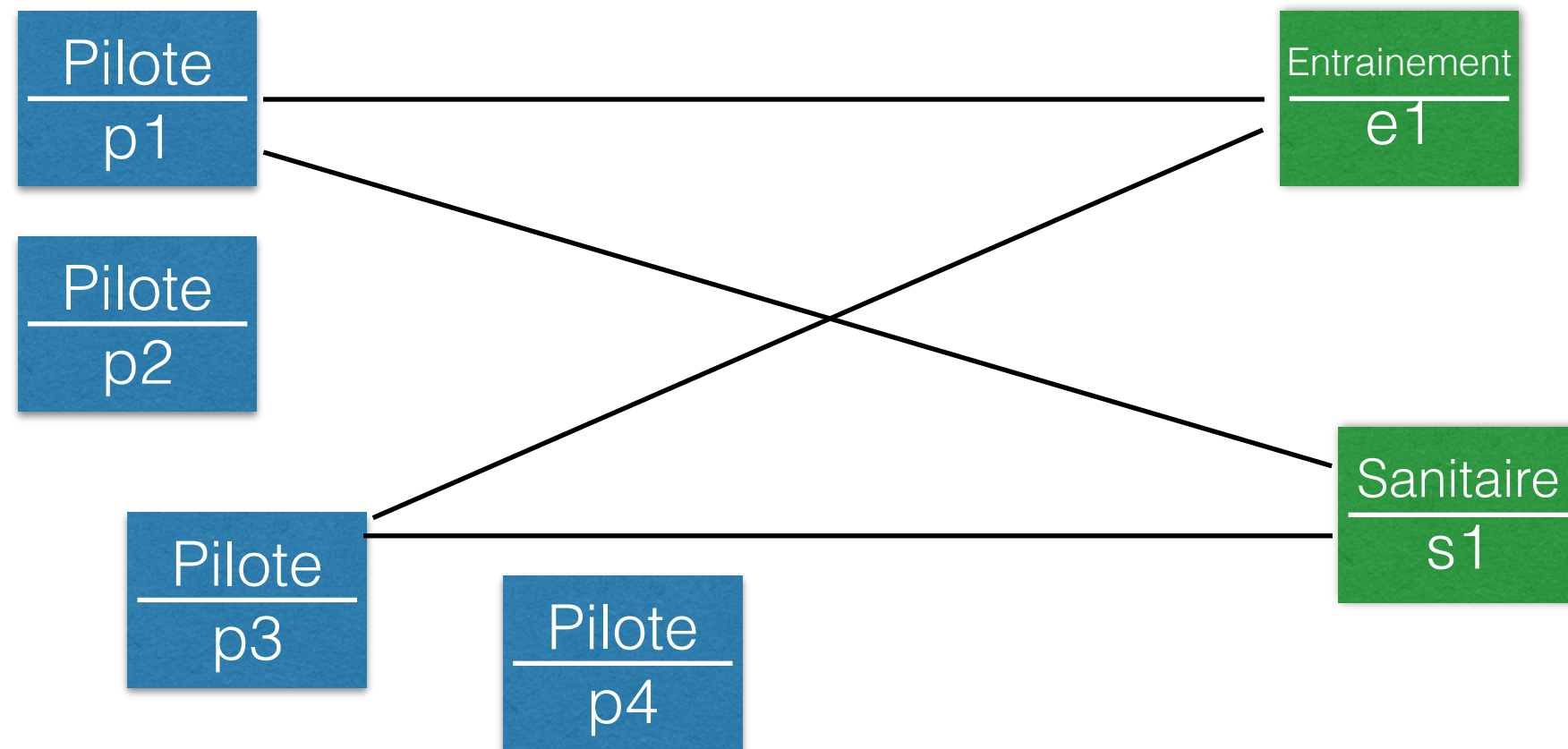
## Contrainte de simultanéité



- Si un objet d'une classe participe à l'une des 2 associations, il participe également à l'autre.
- Remarque: contrainte non proposée à la base dans le formalisme UML, qui permet cependant de définir nos propres contraintes.

# Expression de contraintes

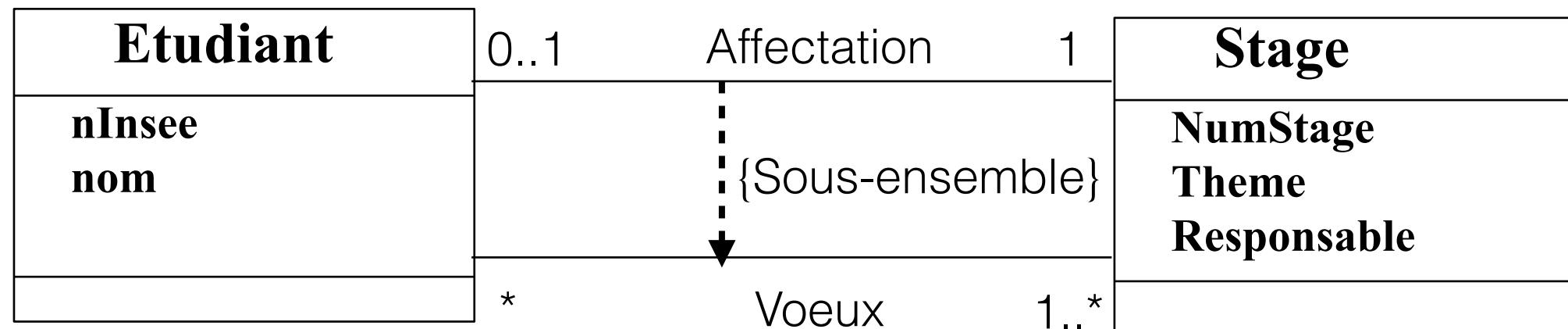
## Contrainte de simultanéité - DO



- Un pilote peut être au repos. En revanche, s'il est affecté à une exercice d'entraînement, il doit aussi être affecté à une mission sanitaire et vice-versa.

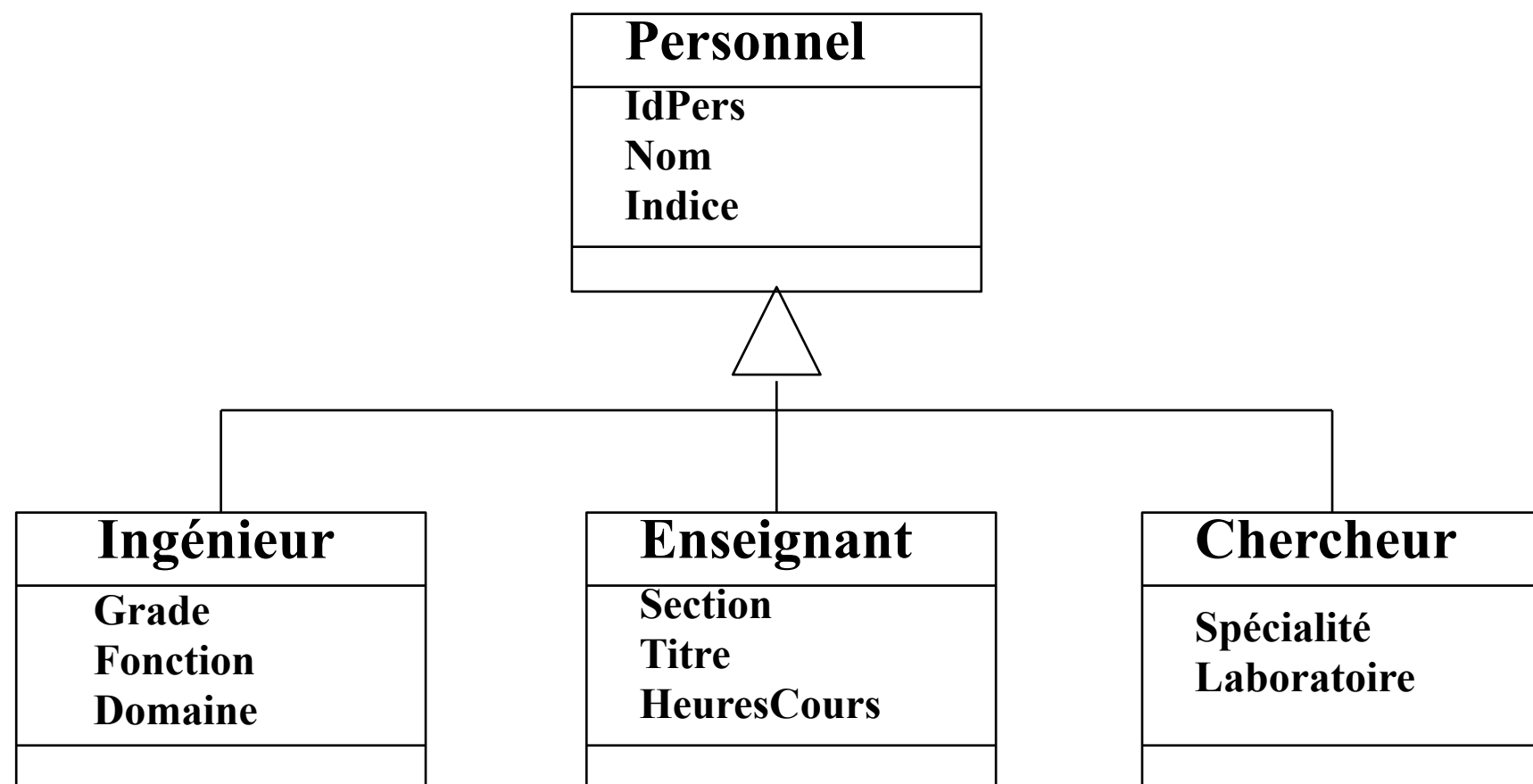
# Expression de contraintes

## Contrainte d'inclusion



- Des étudiants émettent des vœux concernant des stages sachant qu'ils doivent suivre un seul stage.
- Un stage intéresse 0 ou plusieurs étudiants. Pour modéliser le fait que le stage effectué par un étudiant doit être un stage figurant dans les vœux de cet étudiant, il faut ajouter une contrainte d'inclusion

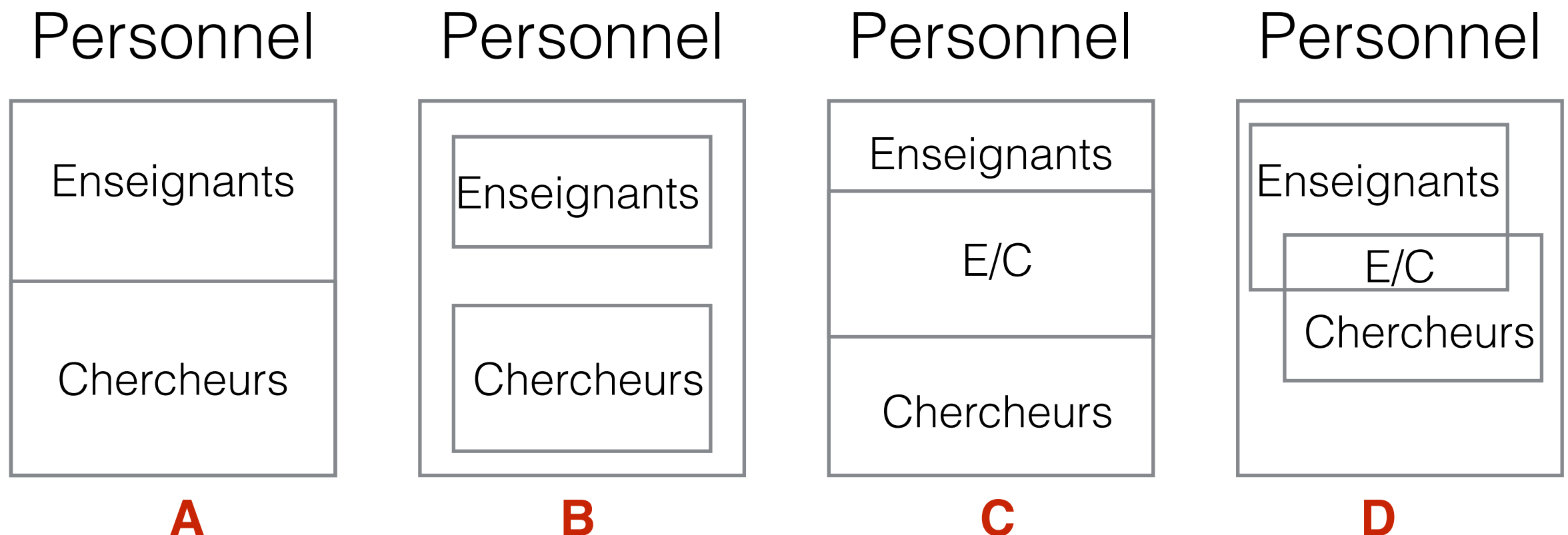
# Généralisation et spécialisation



- Les attributs, les opérations, les relations et les contraintes définies dans les super-classes sont héritées intégralement dans les sous-classes.

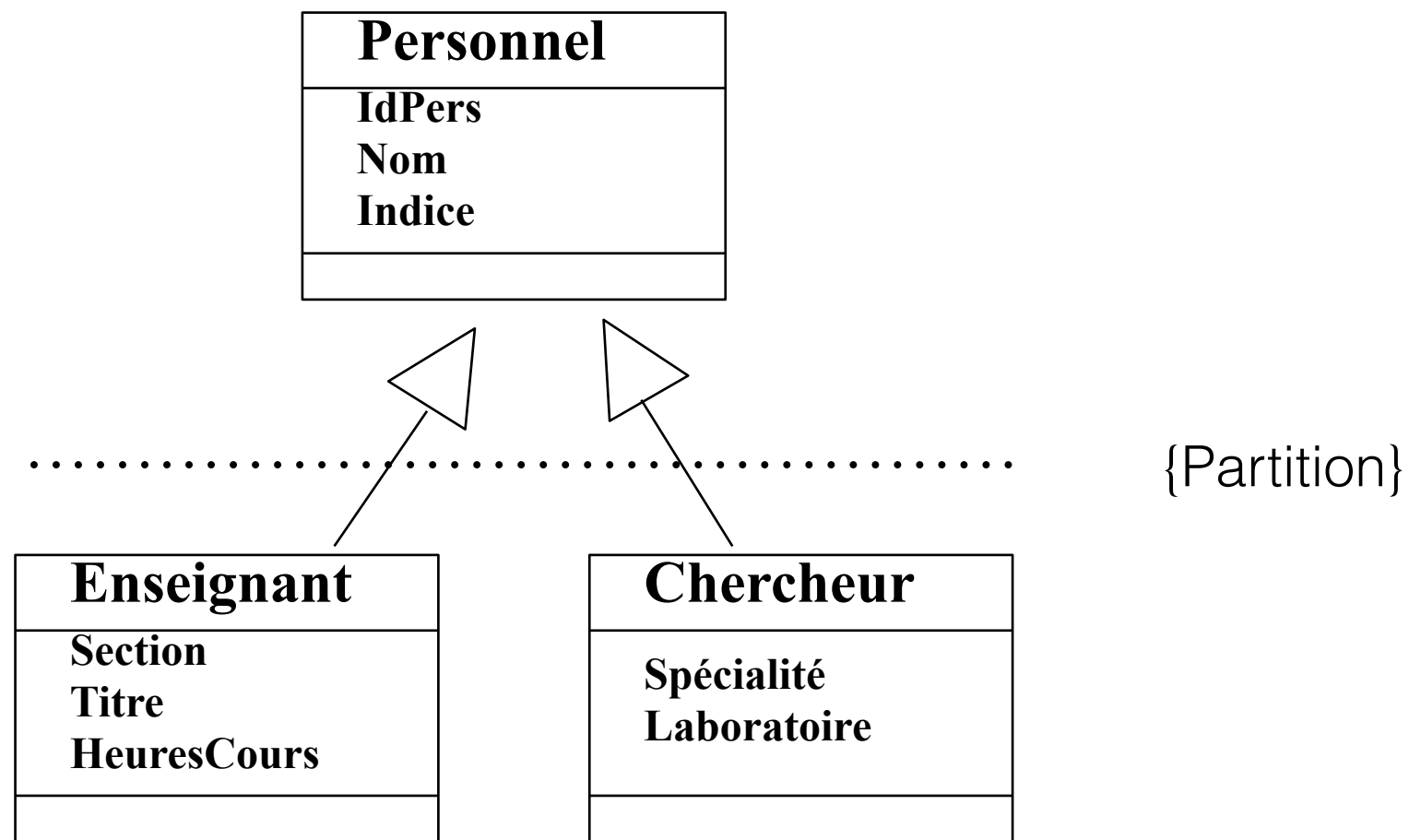
# Différents cas d'héritage

	Couverture	Non couverture
Disjonction	Partition - <span>A</span>	Exclusion - <span>B</span>
Non disjonction	Totalité - <span>C</span>	Absence de contraintes - <span>D</span>



# Héritage avec partition

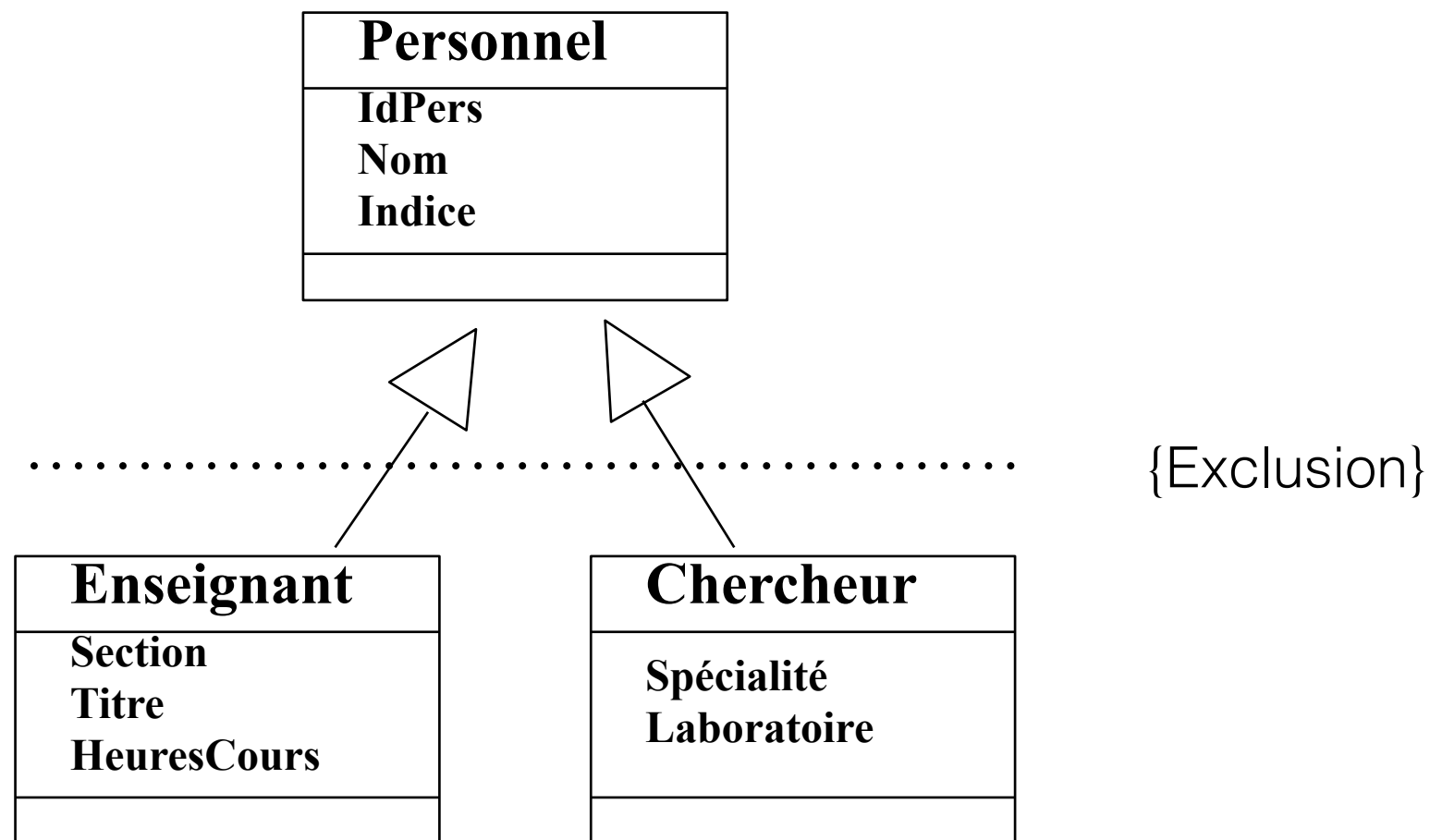
## Disjonction - Couverture (A)



- Le personnel est soit enseignant, soit chercheur.
- L'union des enseignants et des chercheurs constitue tout le personnel

# Héritage avec exclusion

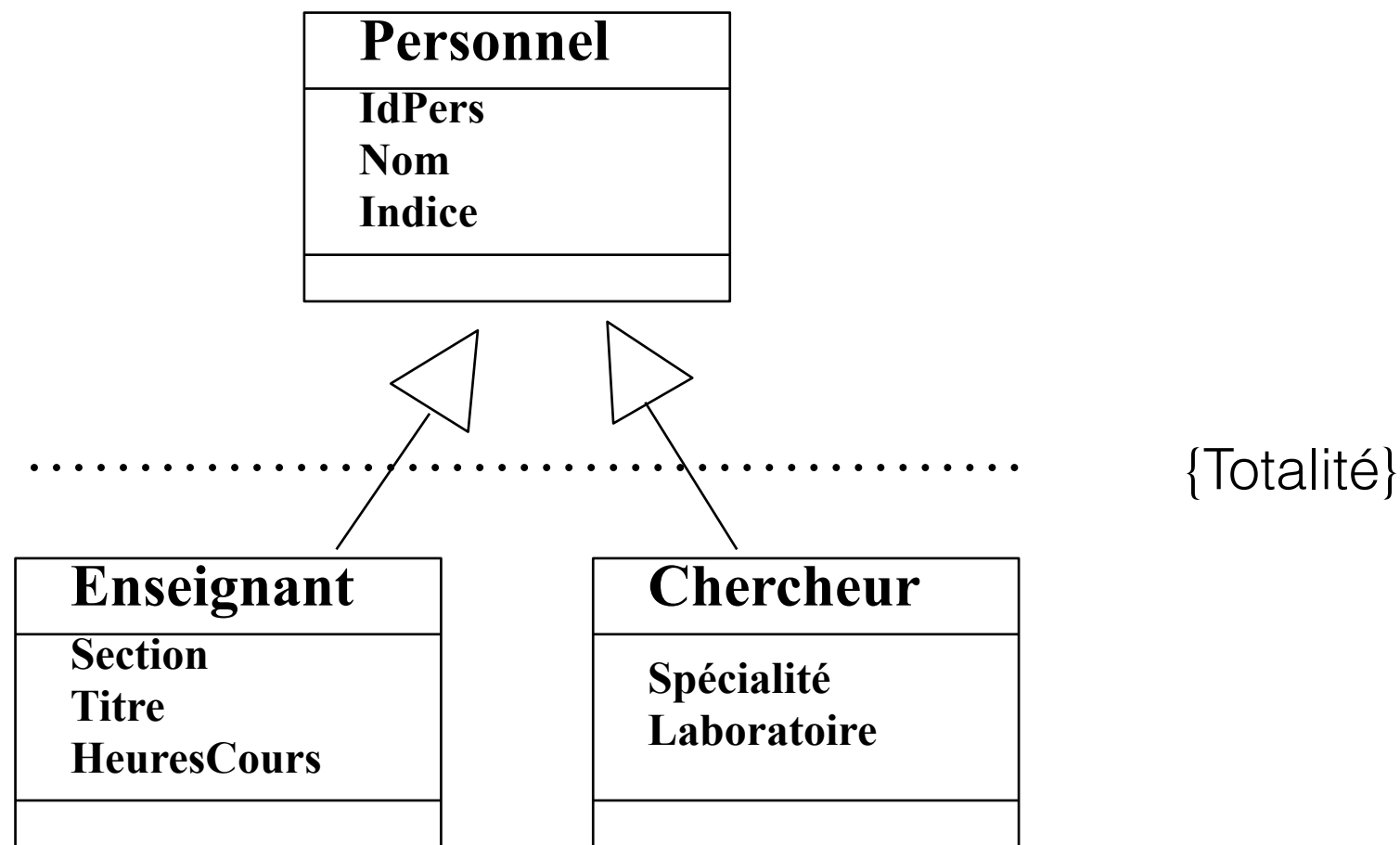
## Disjonction - Non Couverture (B)



- Certains personnels ne sont ni enseignants, ni chercheurs. L'union des enseignants et des chercheurs ne forme pas l'ensemble du personnel.
- Un enseignant ne peut pas être chercheur et vice-versa.

# Héritage avec totalité

## Non Disjonction - Couverture (C)

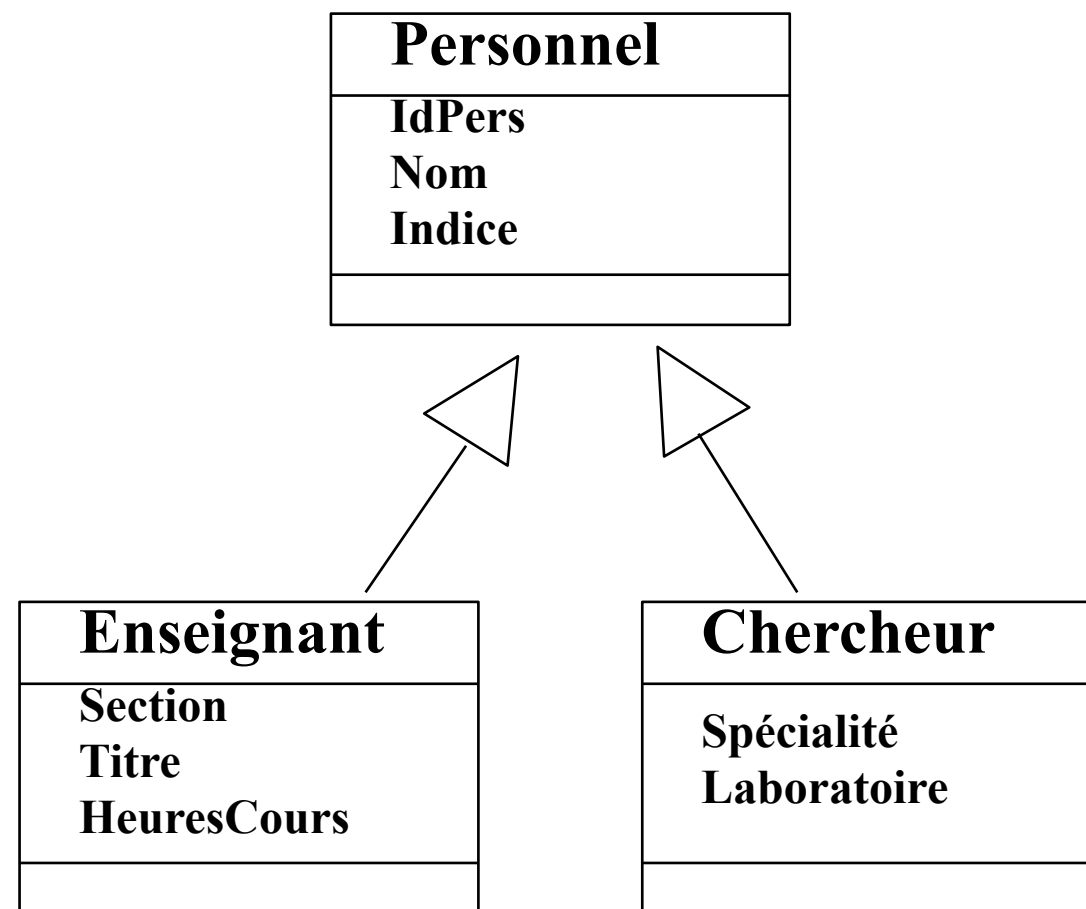


- Il existe des personnels qui sont à la fois enseignants et chercheurs.
- L'union des enseignants, chercheurs et enseignants-chercheurs forme la totalité du Personnel.



# Héritage avec absence de contraintes

## Non Disjonction - Non Couverture (D)



- Cas le plus général : un enseignant peut être Enseignant, Chercheur, Enseignant-Chercheur ou autre chose.

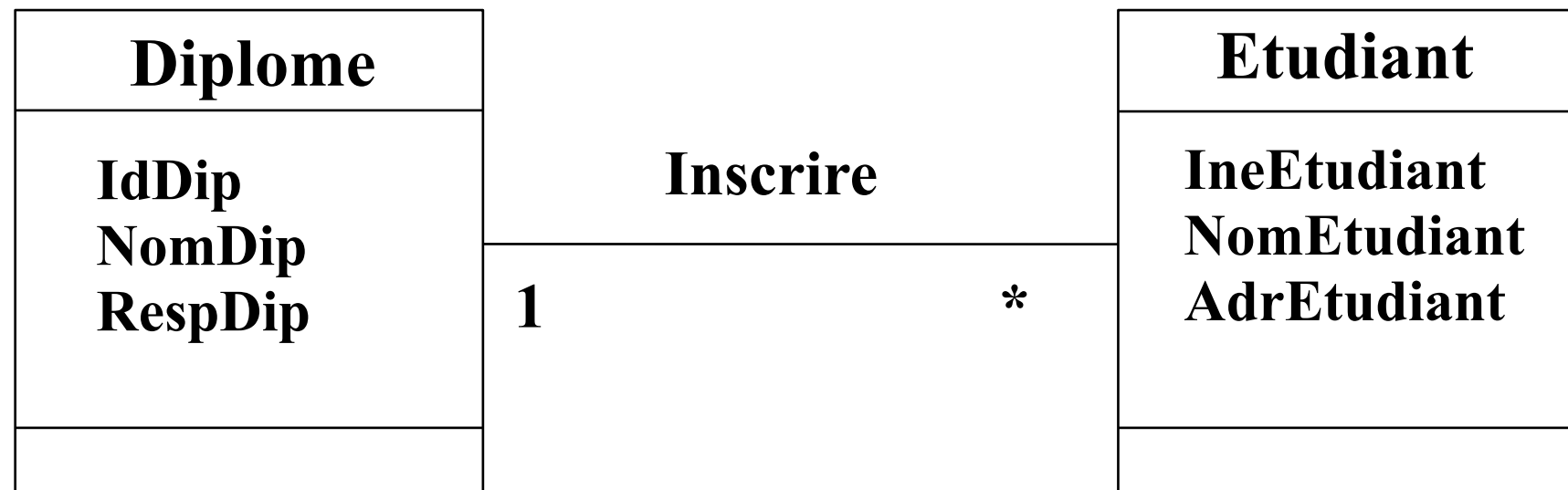
# Passage du conceptuel au logique

## Règles de passage

- **Règle n°1: Classes normales**
  - Chaque classe devient une relation
  - L'identifiant de la classe devient la clé primaire de la relation
- **Règle n°2: Classes d'associations 1-\* (Mère-Fille)**
  - Cette classe disparaît
  - La clé de la relation mère glisse dans la relation fille -> Clé étrangère
- **Règle n°3: Classes d'associations \*-\***
  - Cette classe devient une relation
  - La clé primaire est composée des clés associées (clé primaire composée)
- **Règle n°4: Classes d'associations 1-1**
  - Cas particulier: expliqué plus loin
- **Règle n°5: Héritage**
  - Expliqué plus loin

# Association 1-\*

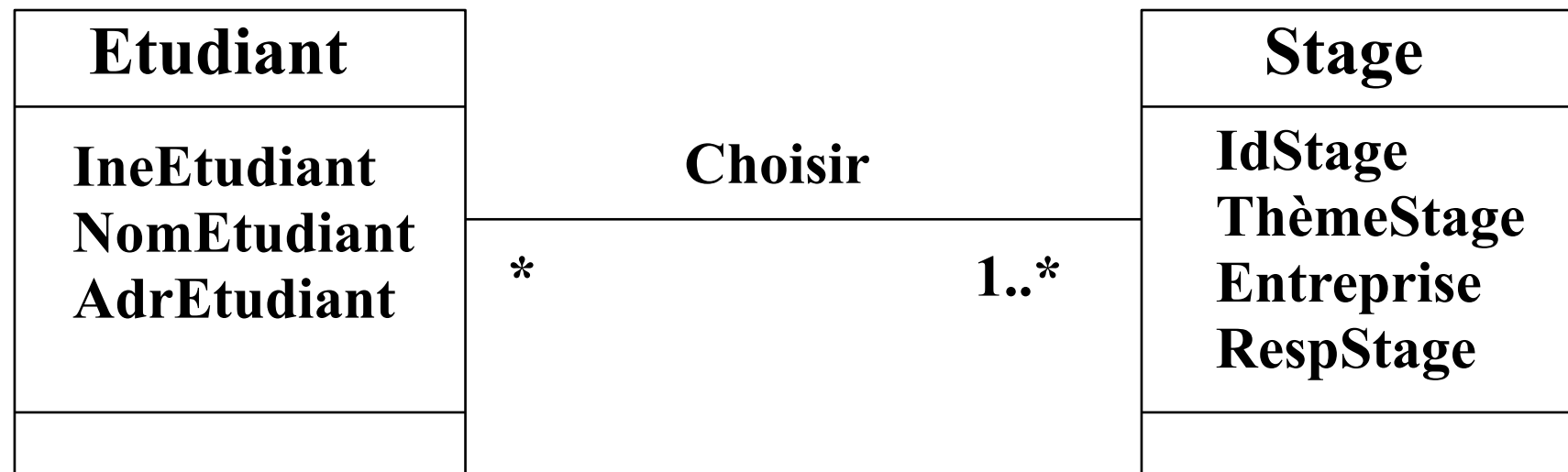
## Exemple



- Diplôme (IdDip, NomDip, RespDip)
- Etudiant (IneEtudiant, NomEtudiant, AdrEtudiant, #IdDip)

# Association \*-\* sans attributs

## Exemple



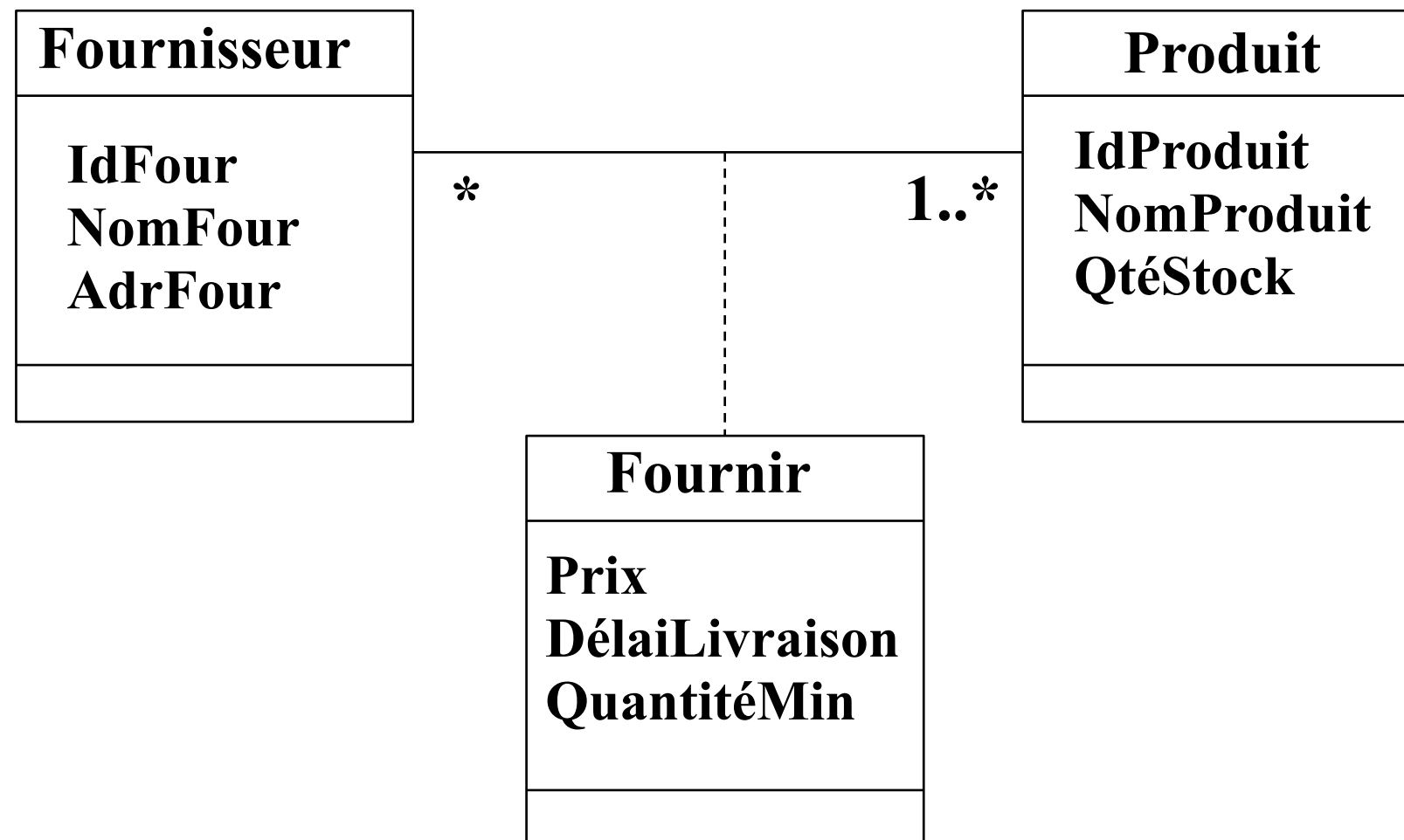
- Etudiant (IneEtudiant, NomEtudiant, AdrEtudiant)
- Choisir (#IneEtudiant, #IdStage)
- Stage (IdStage, ThèmeStage, Entreprise, RespStage)

# Clé primaire composée: postulats

- Autant de composants que de classes associées (N-aires)
- Composée **entièrement** de clés étrangères

# Association \*-\* avec attributs

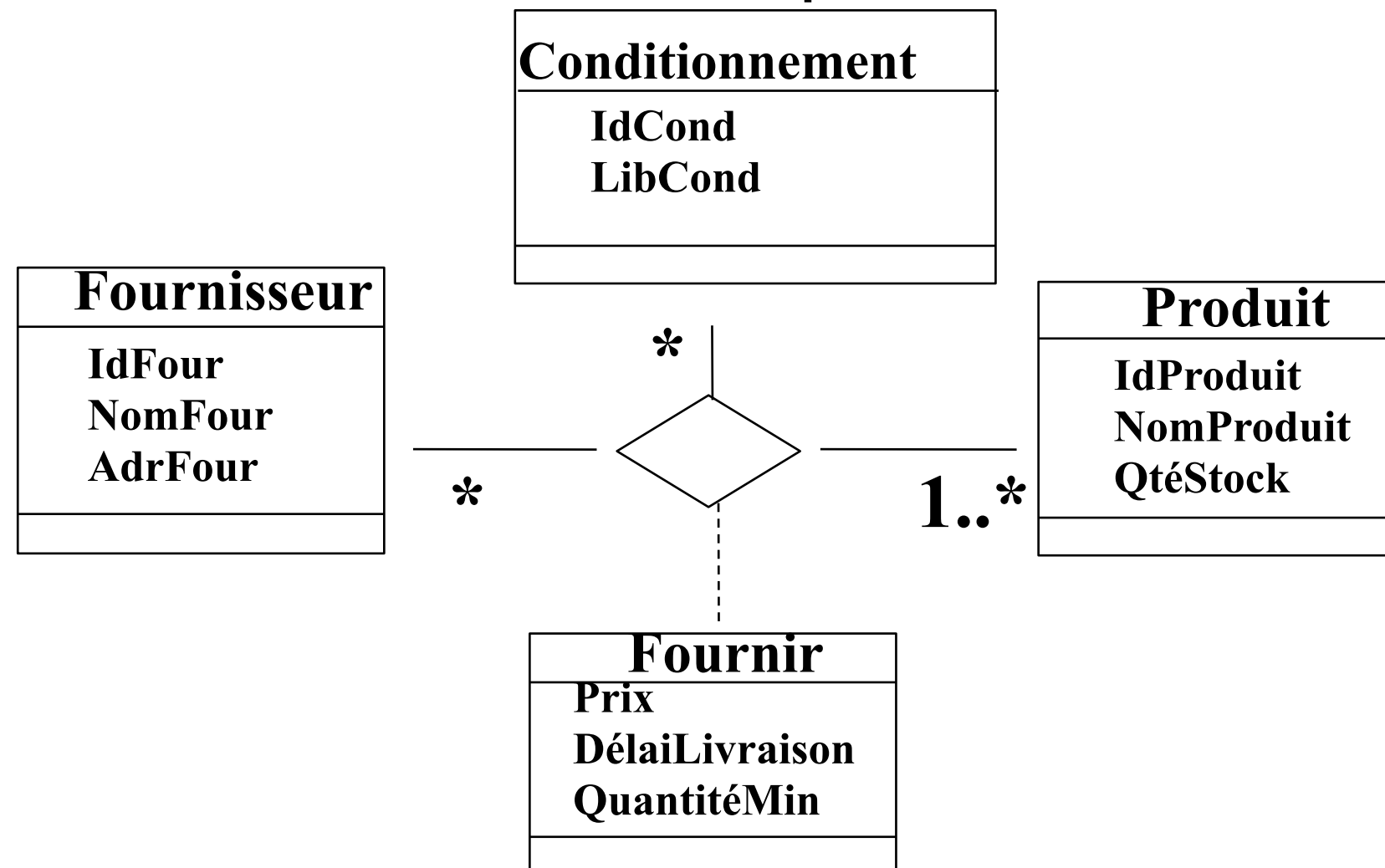
## Exemple



- Fournisseur (IdFour, NomFour, AdrFour)
- Fournir (#IdFour, #IdProduit, Prix, DélaiLivraison, QuantitéMin)
- Produit (IdProduit, NomProduit, QtéStock)

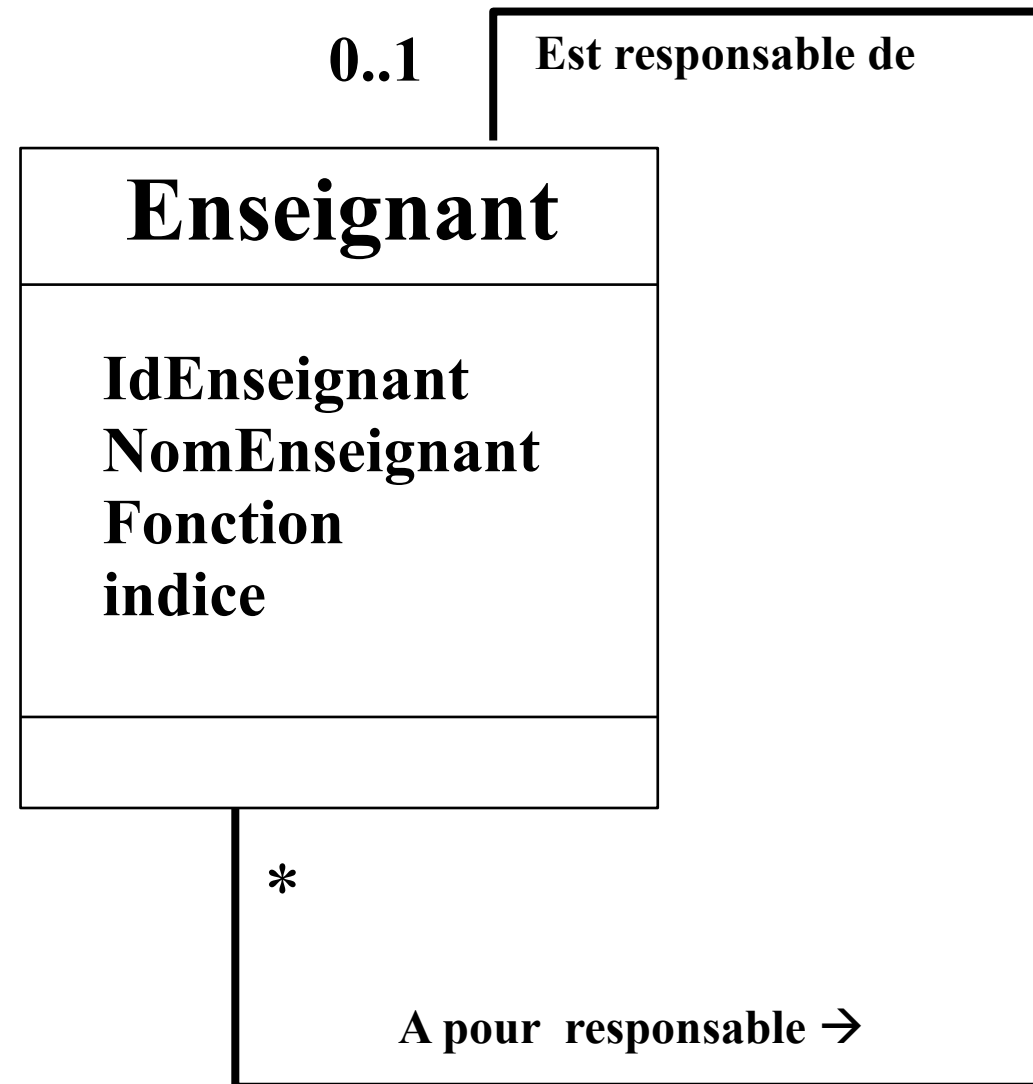
# Associations de type N-aires:

## Exemple



- Fournisseur (IdFour, NomFour, AdrFour)
- Produit (IdProduit, NomProduit, QtéStock)
- Conditionnement (IdCond, LibCond)
- Fournir (#IdFour, #IdProduit, #IdCond, Prix, DélaiLivraison, QuantitéMin)

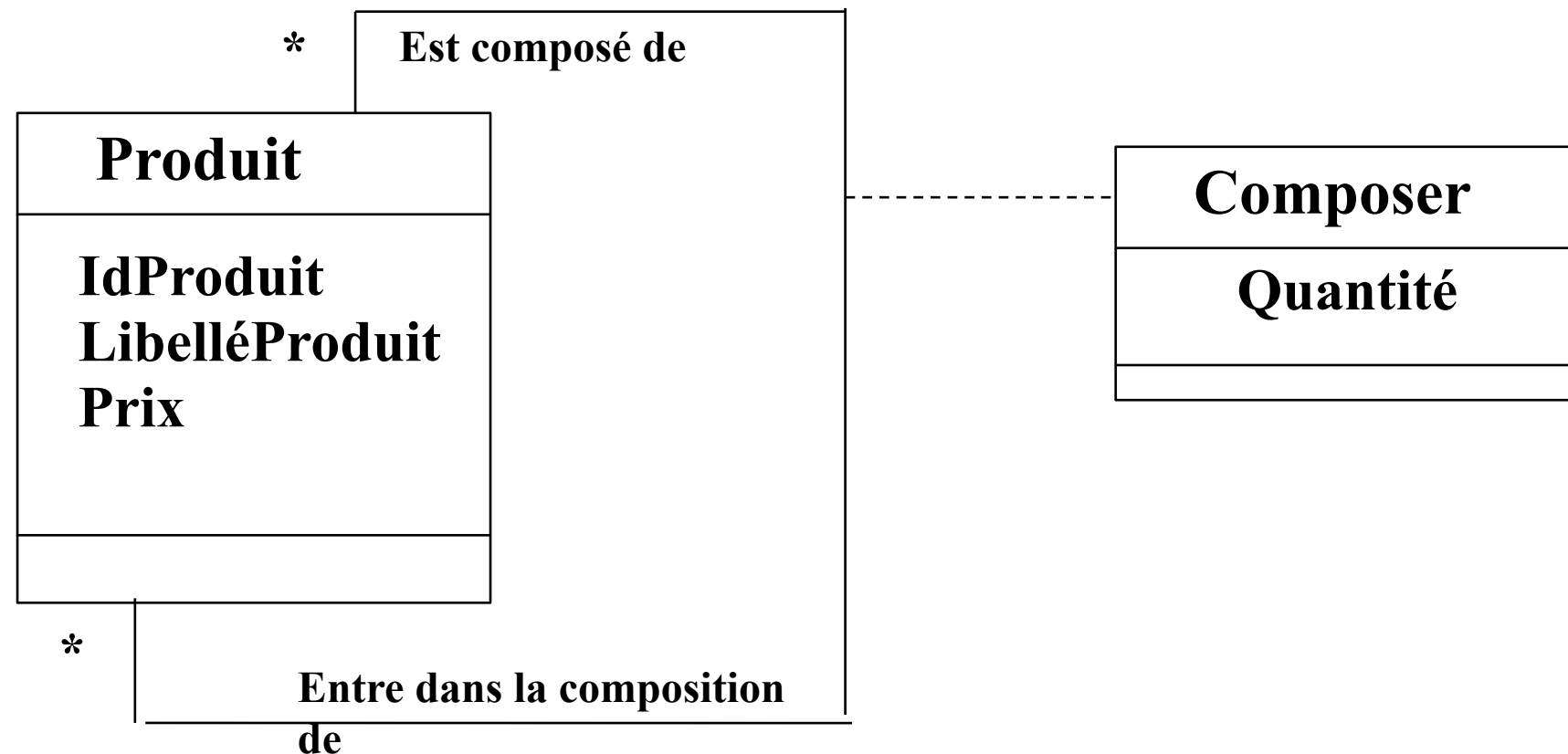
# Association 1-\* réflexive: Exemple



- Enseignant (IdEnseignant, NomEnseignant, Fonction, Indice #IdEns\_Resp)

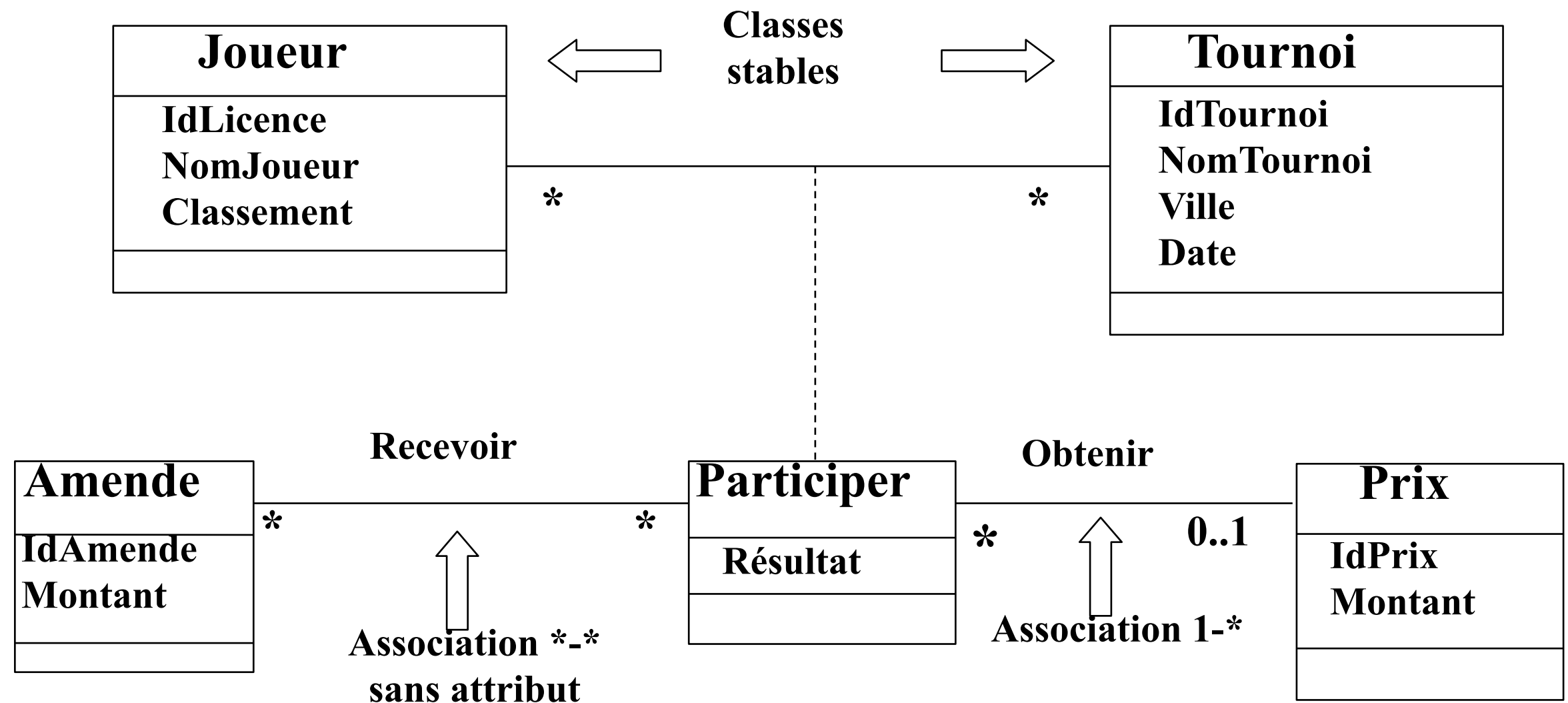


# Association $*-*$ réflexive: Exemple



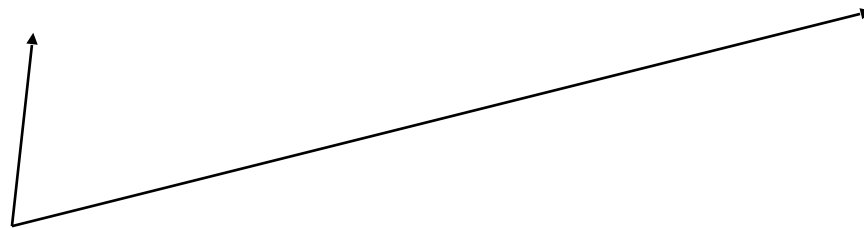
- `Produit (IdProduit, LibelléProduit, Prix)`
- `Composer (#IdProduitComposé, #IdProduitComposant, Quantité)`

# Reprise d'une classe d'association: Exemple



# Reprise d'une classe d'association: Exemple

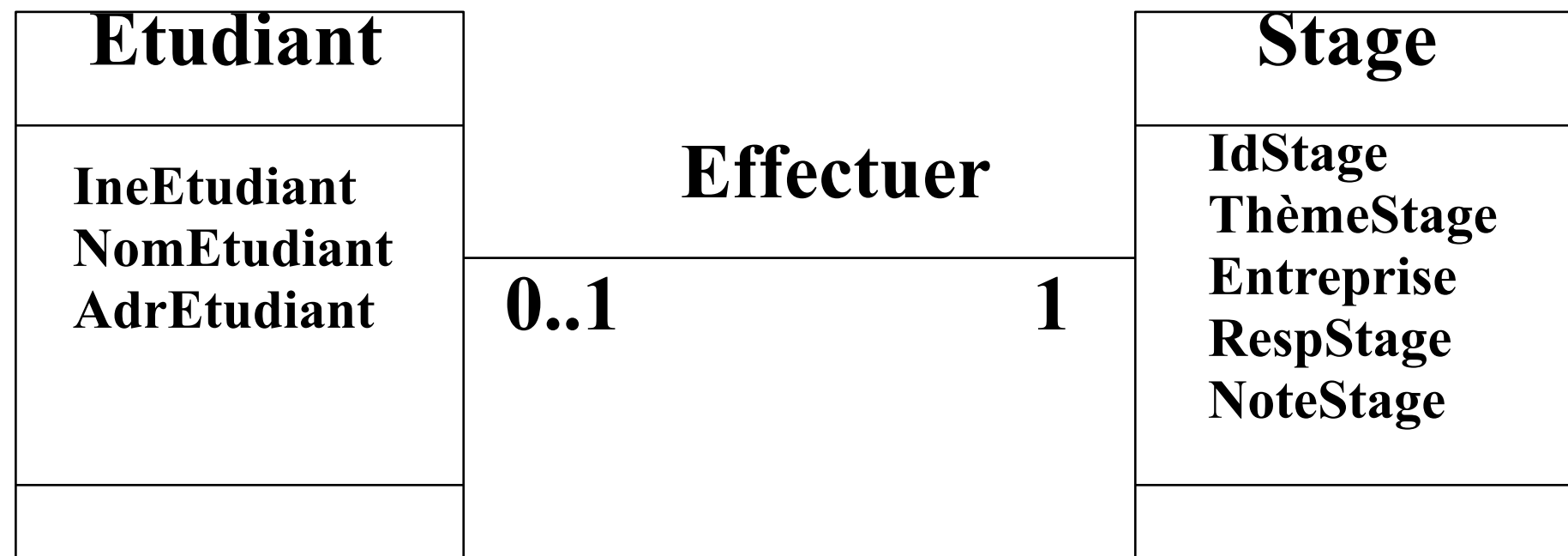
- Joueur (IdLicence, NomJoueur, Classement)
- Tournoi (IdTournoi, NomTournoi, Ville, Date)
- Participer (#IdLicence, #IdTournoi, Résultat, #IdPrix)
- Prix (IdPrix, Montant)
- Amende (IdAmende, Montant)
- Recevoir (#IdAmende, #(IdLicence, IdTournoi))



**2 clés étrangères**

# Associations de type symétrique (1-1)

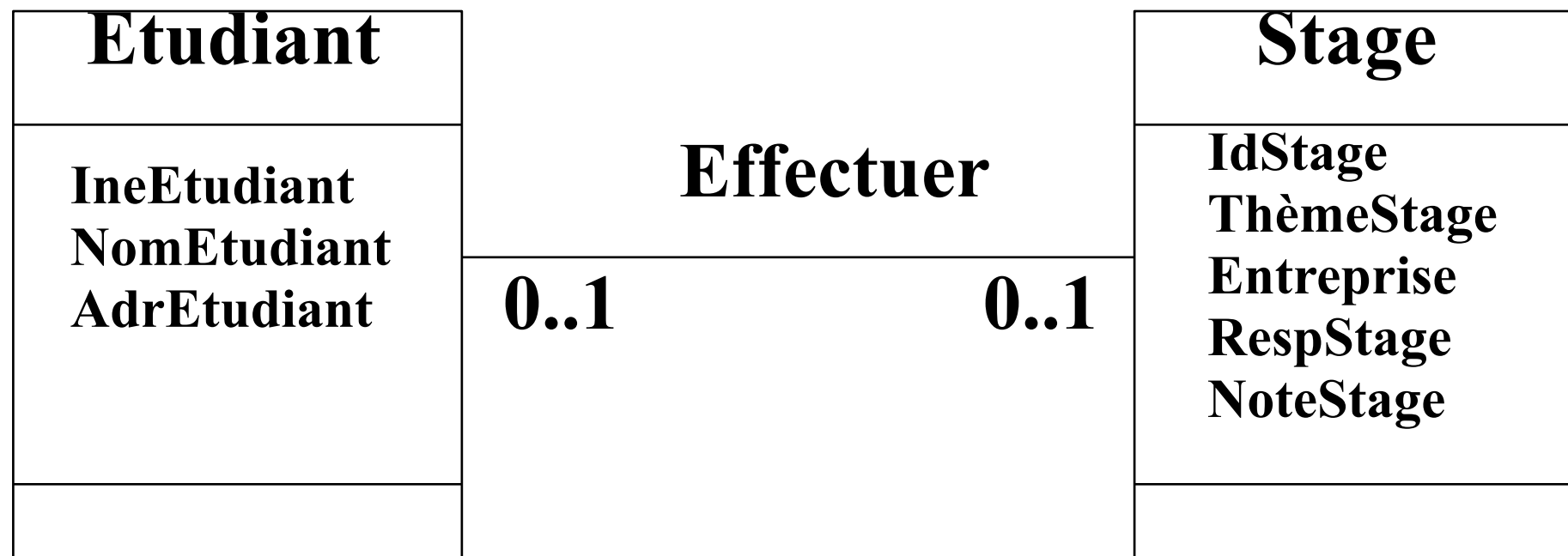
## Exemple



- Etudiant (IneEtudiant, NomEtudiant, AdrEtudiant, #IdStage)
- Stage (IdStage, ThèmeStage, Entreprise, RespStage, NoteStage)

# Associations de type symétrique (1-1)

## Exemple



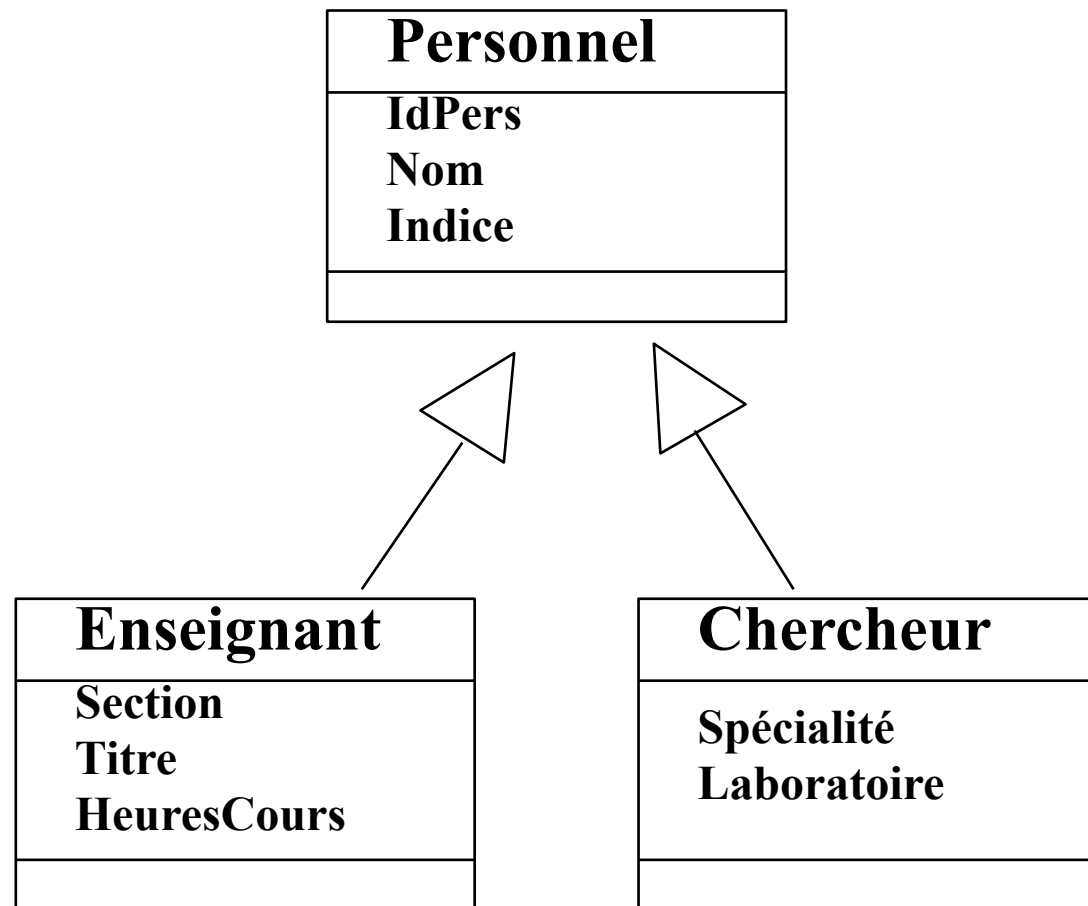
- Etudiant (IneEtudiant, NomEtudiant, AdrEtudiant)
- Stage (IdStage, ThèmeStage, Entreprise, RespStage, NoteStage)
- Effectuer (#IdStage, #IdEtudiant)

# Règle n°5

## Héritage

- Trois familles de décomposition pour traduire une association d'héritage:
  - décomposition par distinction
  - décomposition ascendante (push-down)
  - décomposition ascendante (push-up)

# Décomposition par distinction

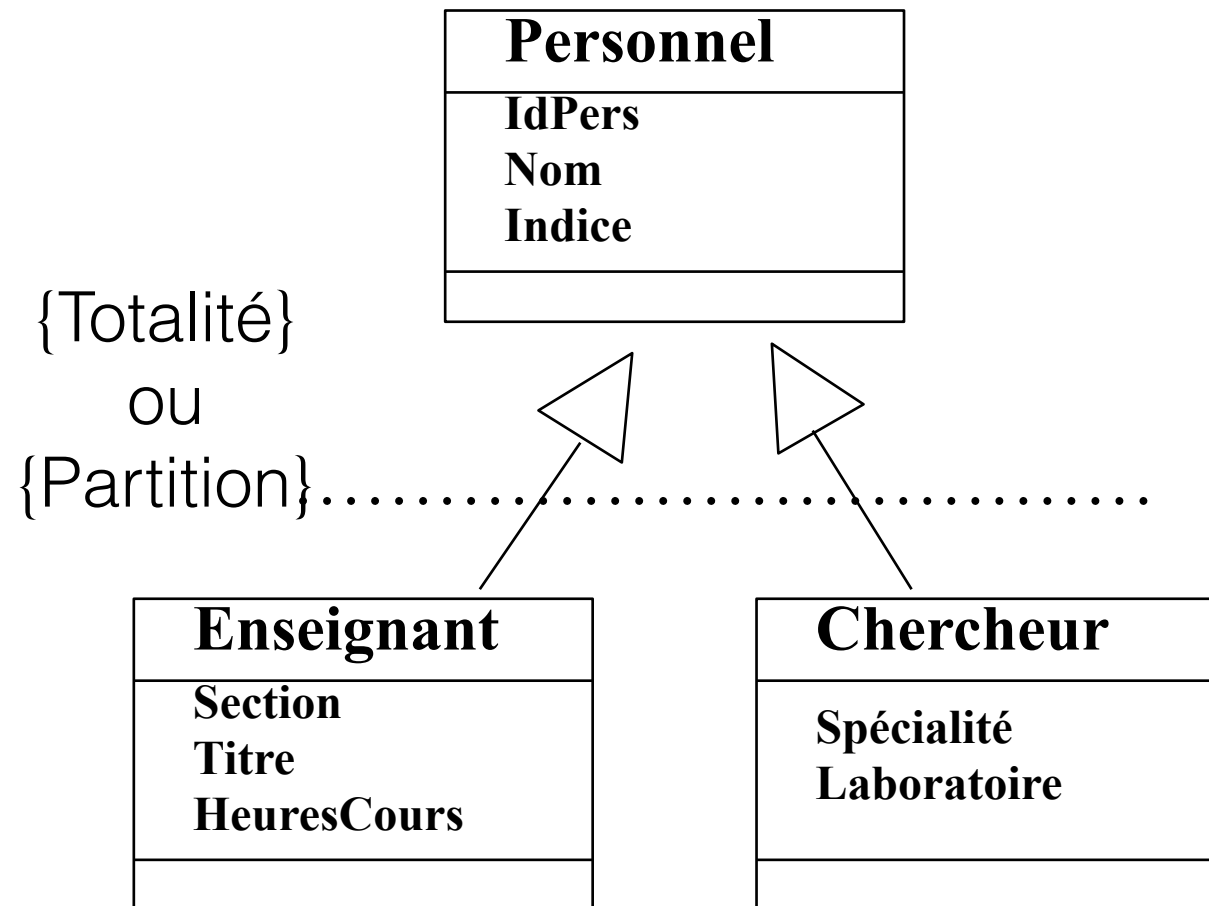


- Enseignant (#IdPers, Section, Titre, HeuresCours)
- Personnel (IdPers, Nom, Indice)
- Chercheur (#IdPers, Spécialité, laboratoire)

- Chaque sous-classe est transformée en relation
- La clé primaire de la relation issue de la sur-classe migre dans la ou les relations issues de la ou des sous-classes et devient à la fois clé primaire et clé étrangère

# Décomposition descendante (push-down)

## Cas 1



- Enseignant (IdPers, Nom, indice, Section, Titre, HeuresCours)

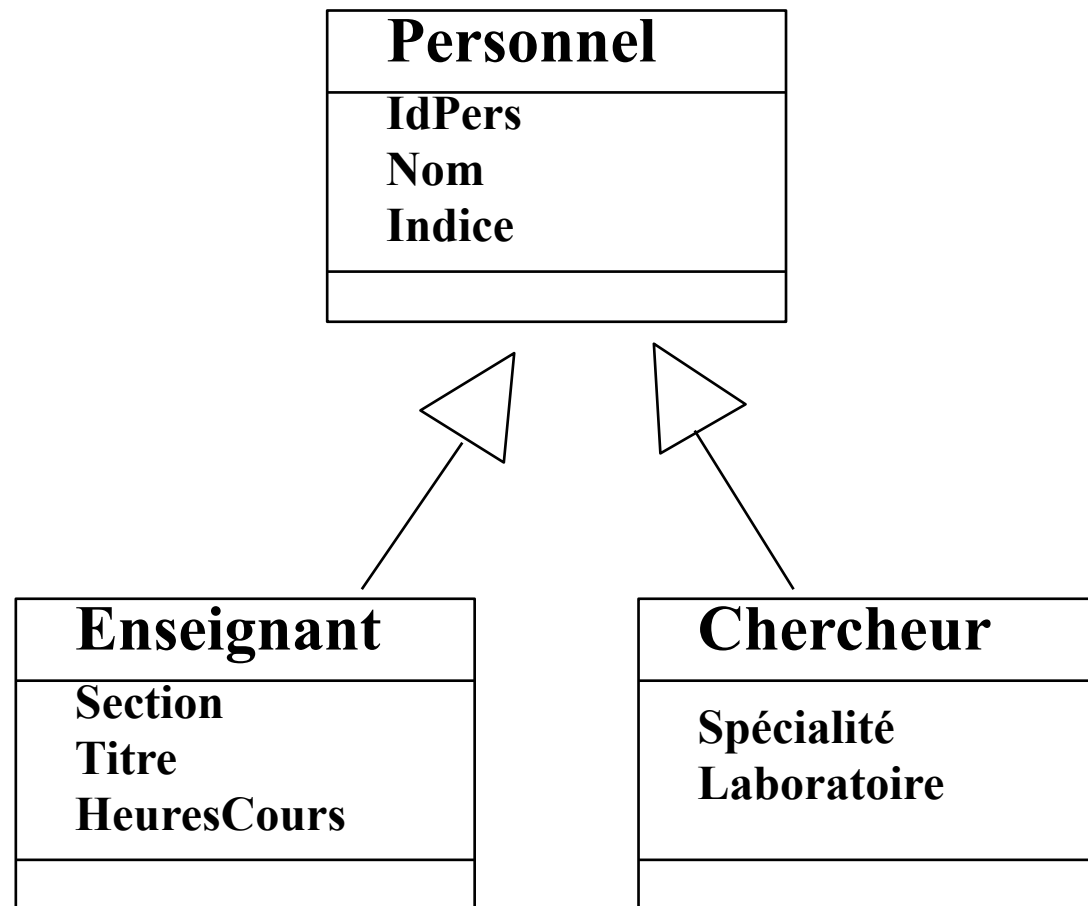
- Chercheur (IdPers, Nom, Indice, Spécialité, laboratoire)

- S'il existe une contrainte de totalité ou de partition sur l'association, il est possible de ne pas traduire la relation issue de la sur-classe.
- Il faut alors migrer tous ses attributs dans la ou les relations issues de la ou des sous-classes



# Décomposition descendante (push-down)

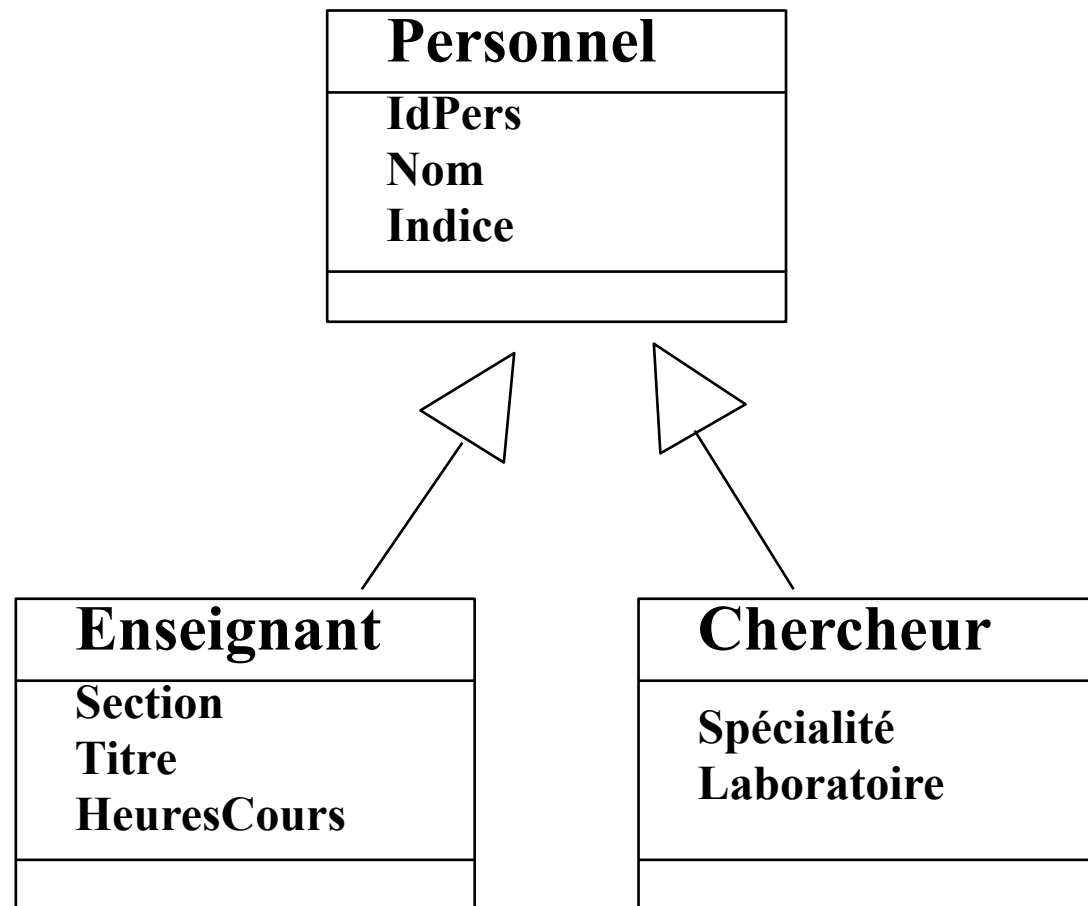
## Cas 2



- Enseignant (IdPers, Nom, indice, Section, Titre, HeuresCours)
- Chercheur (IdPers, Nom, Indice, Spécialité, laboratoire)
- Personnel (Idpers, Nom, Indice)

- Il faut migrer les attributs de la sur-classe dans les sous-classes et on garde la sur-classe comme relation.

# Décomposition ascendante (push-up)



- Personnel (Idpers, Nom, Indice, Section, Titre, HeuresCours, Spécialité, Laboratoire)

- Il faut supprimer la ou les relations issues de la ou les sous-classes et faire migrer les attributs dans la relation issue de la sur-classe.

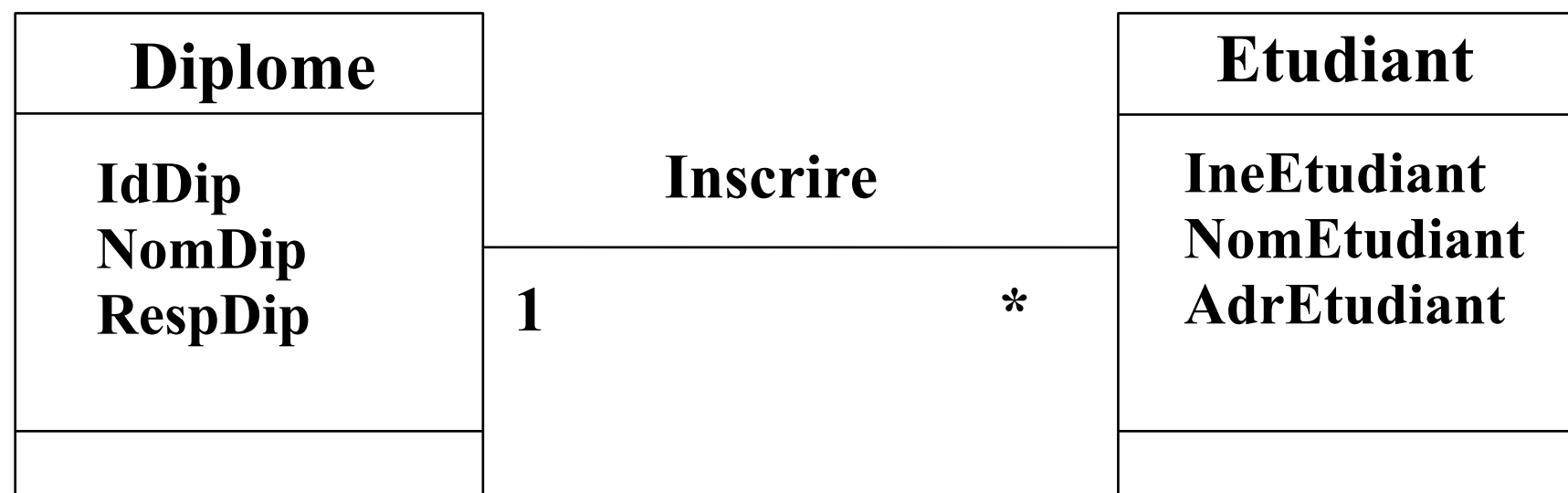
# Une fois le schéma relationnel fait...

- Beaucoup de contraintes (de cardinalité ou autres) n'ont pas été traduites
- On achève de les traduire au niveau du schéma physique
- C'est à dire au niveau de la création des tables et des contraintes associées
- Utilisation du langage SQL

# Schéma physique

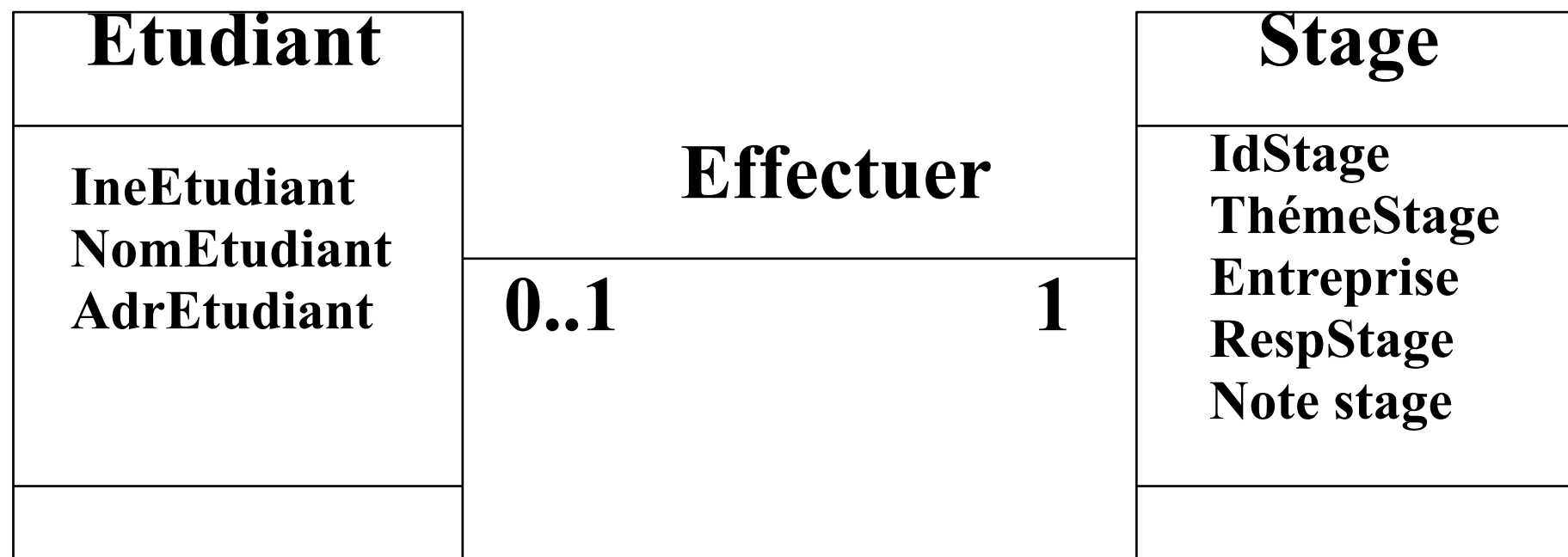
- Les clés primaires et étrangères se traduisent par des contraintes sur les tables associées
- Reste à prendre en compte:
  - la traduction de certaines cardinalités
  - les associations d'agrégation et de composition
  - la traduction des contraintes de partition, exclusion, totalité, inclusion,
  - l'héritage et ses contraintes

# Cardinalité minimale 1 dans le cas d'association 1-\*



- Diplôme (IdDip, NomDip, RespDip)
- Etudiant (IneEtudiant, NomEtudiant, AdrEtudiant, #IdDip)

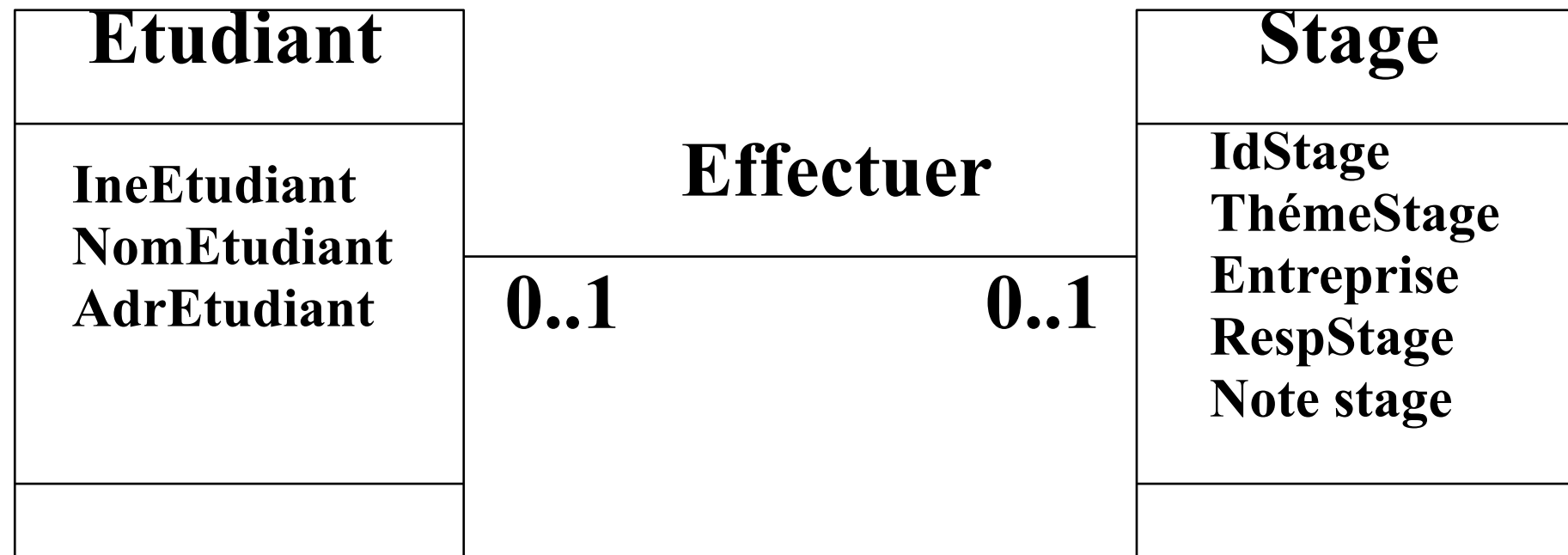
# Associations de type symétrique (1-1)



- Etudiant (IneEtudiant, NomEtudiant, AdrEtudiant, #IdStage)
- Stage (IdStage, ThèmeStage, Entreprise)

**Contraintes**  
**not null**  
**+ Unique**

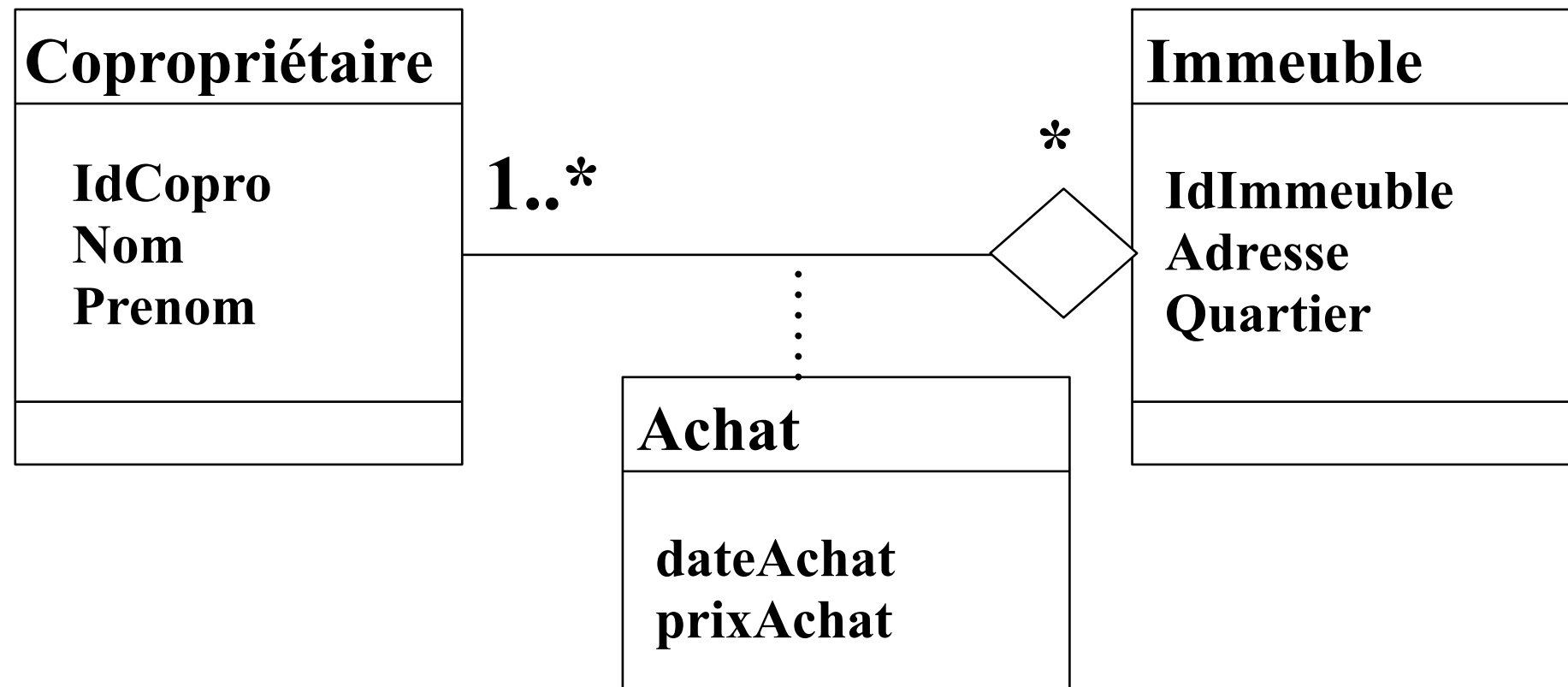
# Associations de type symétrique (1-1)



- Etudiant (IneEtudiant, NomEtudiant, AdrEtudiant)
- Stage (IdStage, ThèmeStage, Entreprise)
- Effectuer (#IdStage, #IdEtudiant)

**Unique**

# Transformation des associations d'agrégation



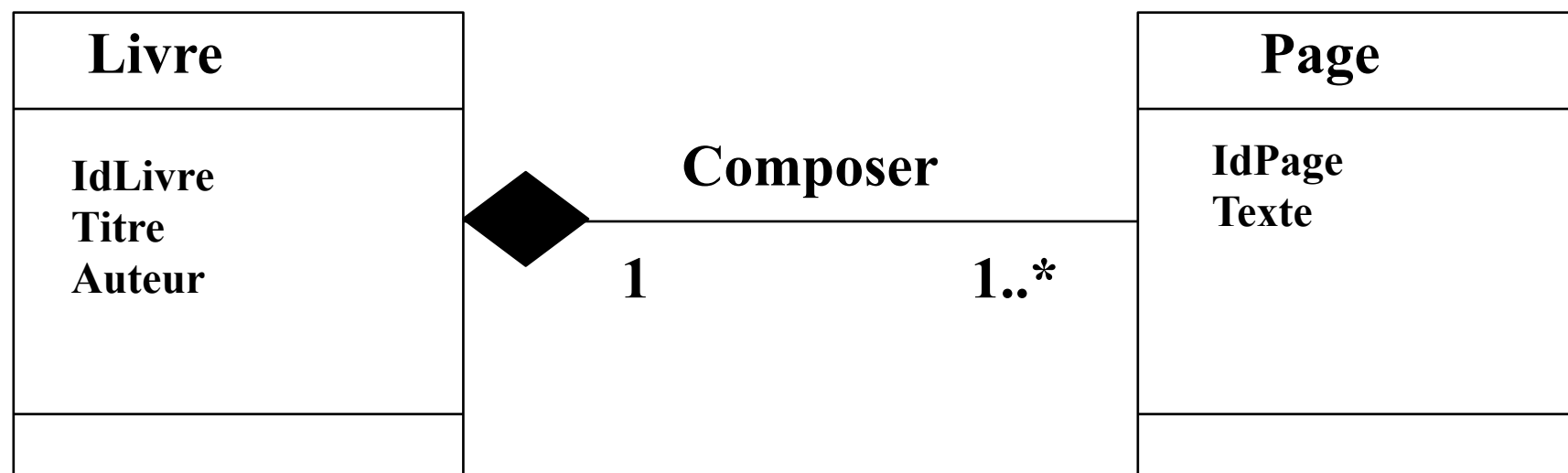
La suppression d'un immeuble doit se répercuter sur la suppression des achats liés à cet immeuble.

- Copropriétaire (IdCopro, nom, prenom)
- Immeuble (IdImmeuble, adresse, quartier)
- Achat (#idCopro, #IdImmeuble, dateAchat, prixAchat)

**Clause « on delete cascade »**



# Transformation des associations de composition



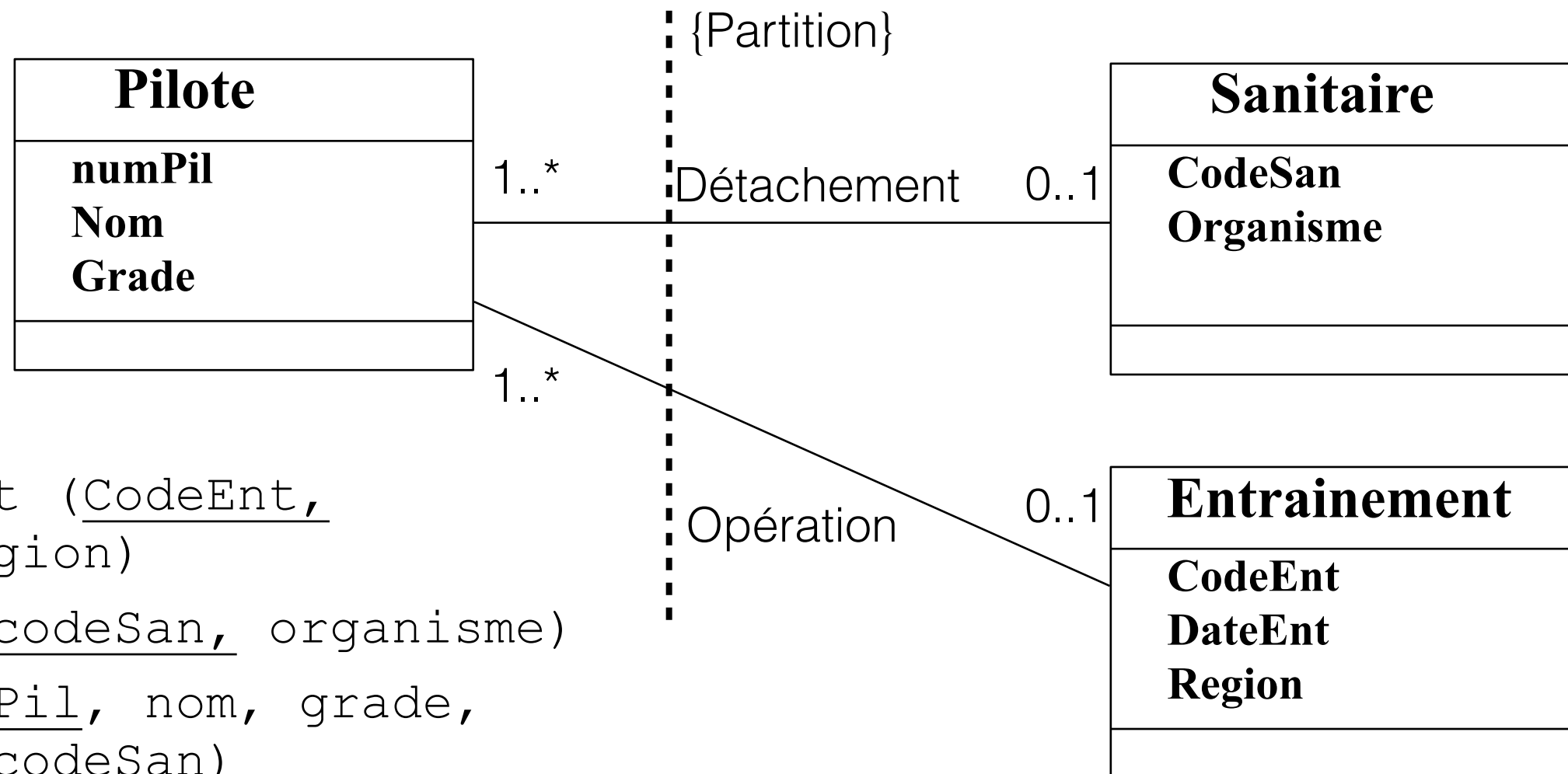
La suppression d'un livre entraine la suppression des pages qu'il contient.

- Livre (IdLivre, titre, auteur)
- Page (IdPage, texte, #idLivre)

**Clause « on delete cascade »**

# Traduction des contraintes

## Partition

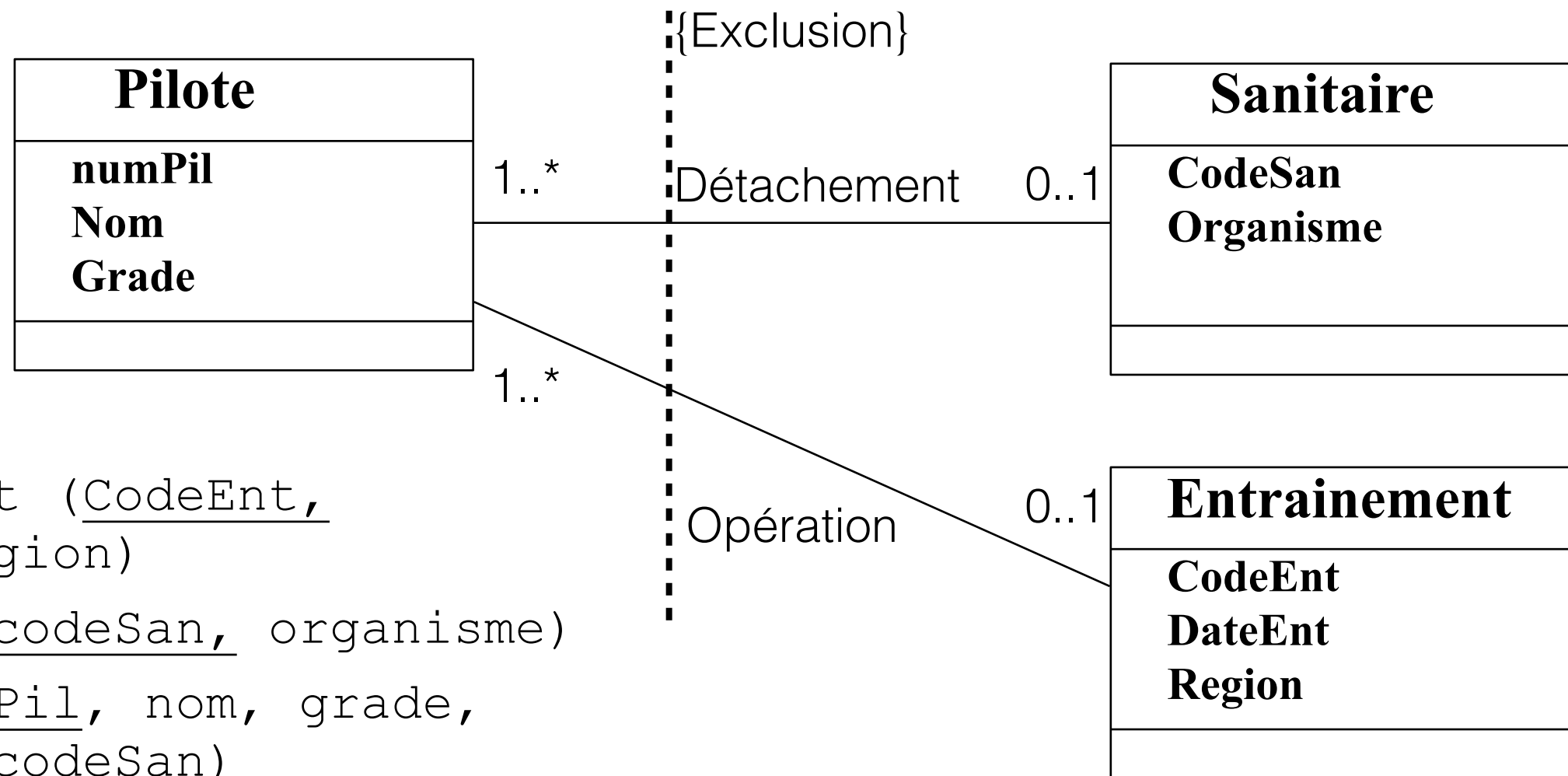


- Entraînement (CodeEnt, dateEnt, region)
- Sanitaire (codeSan, organisme)
- Pilote (numPil, nom, grade, #codeEnt, #codeSan)

**Contrainte check sur la table Pilote :**  
**((codeSan is null and codeEnt is not null)**  
**OR**  
**(codeSan is not null and CodeEnt is null))**

# Traduction des contraintes

## Exclusion

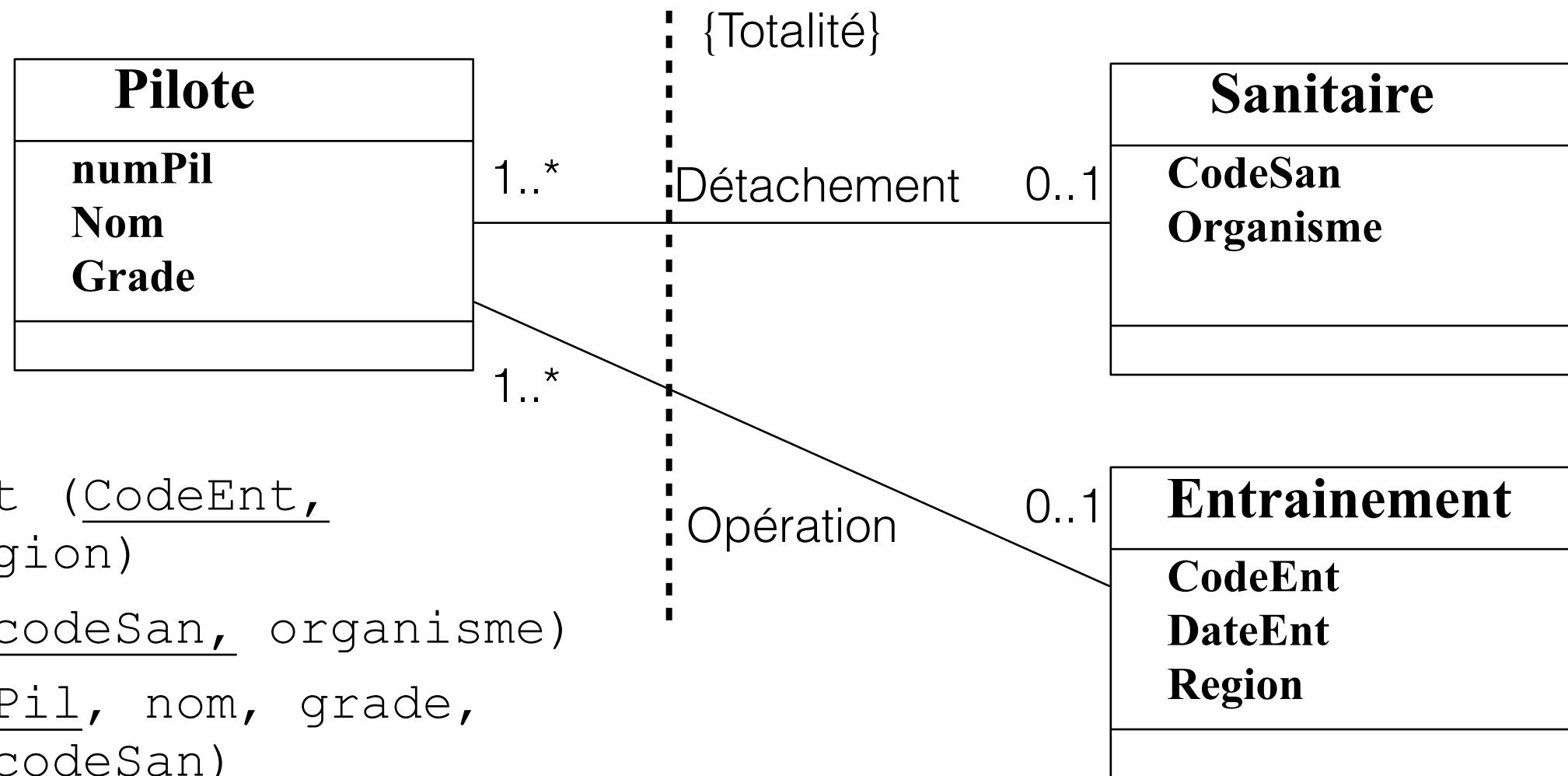


- Entrainement (CodeEnt, dateEnt, region)
- Sanitaire (codeSan, organisme)
- Pilote (numPil, nom, grade, #codeEnt, #codeSan)

**Contrainte check sur la table Pilote :**  
**(codeSan is null or codeEnt is null)**

# Traduction des contraintes

## Totalité

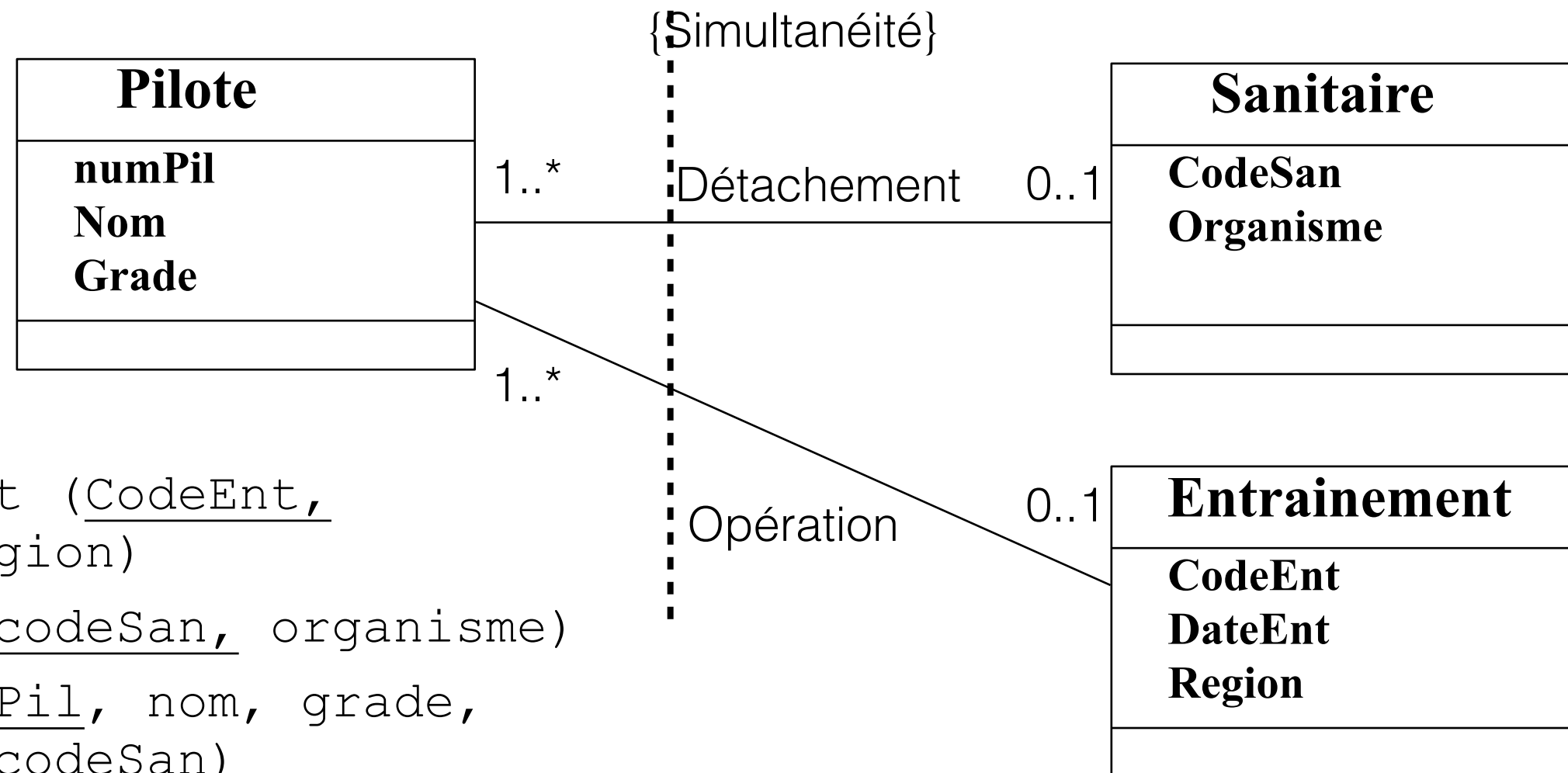


- Entraînement (CodeEnt, dateEnt, region)
- Sanitaire (codeSan, organisme)
- Pilote (numPil, nom, grade, #codeEnt, #codeSan)

**Contrainte check sur la table Pilote :**  
**(codeSan is not null or codeEnt is not null)**

# Traduction des contraintes

## Simultanéité

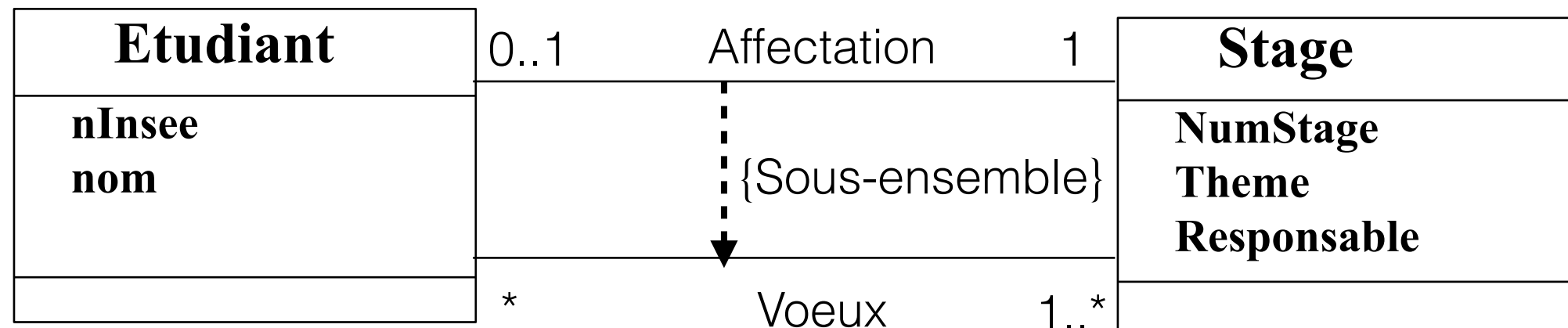


- Entraînement (CodeEnt, dateEnt, region)
- Sanitaire (codeSan, organisme)
- Pilote (numPil, nom, grade, #codeEnt, #codeSan)

**Contrainte check sur la table Pilote :**  
**((codeSan is null and codeEnt is null)**  
**OR**  
**(codeSan is not null and CodeEnt is not null))**

# Traduction de contraintes

## Inclusion



- Etudiant (NInsee, , nom, #numSta)
- Stage (NumStage, thème, responsable)
- Voeux (#NInsee, #NumStage)

**Alter table Etudiant add  
constraint fk\_inclusion  
foreign key (ninsee, numsta) references Voeux (ninsee, numsta);**

# Héritage

## Rappel des différents cas

	Distinction	Descendante	Ascendante
<b>Partition</b>		Enseignant ( <u>IdPers</u> , Nom, indice, Section, Titre, HeuresCours)	
<b>Totalité</b>	Enseignant (# <u>IdPers</u> , Section, Titre, HeuresCours)	Chercheur ( <u>IdPers</u> , Nom, Indice, Spécialité, laboratoire)	
<b>Exclusion</b>	Personnel ( <u>IdPers</u> , Nom, Indice)	Enseignant ( <u>IdPers</u> , Nom, indice, Section, Titre, HeuresCours)	Personnel ( <u>Idpers</u> , Nom, Indice, Section, Titre, HeuresCours, Spécialité, Laboratoire)
<b>Sans contrainte</b>	Chercheur (# <u>IdPers</u> , Spécialité, laboratoire)	Chercheur ( <u>IdPers</u> , Nom, Indice, Spécialité, laboratoire)	
		Personnel ( <u>Idpers</u> , Nom, Indice)	

# Traduction des contraintes

- **Contrainte de partition (A+B)**

- Contrainte A: il n'existe aucun personnel pouvant être à la fois Enseignant et Chercheur
- Contrainte B: il n'existe pas de personnel n'étant ni Enseignant ni Chercheur

- **Contrainte de totalité (B+C)**

- Contrainte C: un personnel peut être à la fois Enseignant et Chercheur

- **Contrainte d'exclusion (A+D)**

- Contrainte D: un personnel peut n'être ni Enseignant ni Chercheur

- **Sans contrainte (C+D)**



# Traduction des contraintes

- Quelle que soit la décomposition (distinction, ascendante, descendante) il faut traduire les contraintes liées au type d'héritage
- A titre d'exemple, on regarde ce qu'il faudrait faire sur la décomposition par distinction

# Décomposition par distinction

## Contrainte A

- Déclencheur sur Enseignant

```
CREATE OR REPLACE TRIGGER T_B_IU_Enseignant
BEFORE INSERT OR UPDATE OF idPers ON Enseignant
FOR EACH ROW
DECLARE
    id Enseignant.idPers%TYPE;
BEGIN
    SELECT idPers INTO id
    FROM Chercheur WHERE idPers = : NEW.idPers;
    RAISE_APPLICATION_ERROR ( -20001, ' Le Personnel ' ||
    TO_CHAR(id) || 'est déjà Chercheur...' );
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        NULL;
END;
/
```

# Décomposition par distinction

## Contrainte A

- Déclencheur sur Chercheur

```
CREATE OR REPLACE TRIGGER T_B_IU_Chercheur
BEFORE INSERT OR UPDATE OF idPers ON Chercheur
FOR EACH ROW
DECLARE
    id Chercheur.idPers%TYPE;
BEGIN
    SELECT idPers INTO id
    FROM Enseignant WHERE idPers = : NEW.idPers;
    RAISE_APPLICATION_ERROR ( -20001, ' Le Personnel ' ||
    TO_CHAR(id) || ' est déjà Enseignant...' );
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        NULL;
END;
/
```

# Décomposition par distinction

## Contrainte B

- Insertion de données: 2 procédures

– Ajout d'un enseignant

```
CREATE PROCEDURE ajout_enseignant (pIdPers Personne.idPers%TYPE, pnom
  Personne.nom%TYPE, pIndice Personne.Indice%TYPE, pTitre
  Enseignant.titre%TYPE, pHeuresCours Enseignant.HeuresCours%TYPE) as
BEGIN
INSERT INTO Personnel VALUES (pidPers, pnom, pIndice);
INSERT INTO Enseignant VALUES (pidPers, ptitre, pHeuresCours);
END;
/
```

– Ajout d'un chercheur

```
CREATE PROCEDURE ajout_chercheur (pIdPers Personne.idPers%TYPE, pnom
  Personne.nom%TYPE, pIndice Personne.Indice%TYPE, pSpecialite
  Chercheur.Specialite%TYPE, plaboratoire Chercheur.laboratoire%TYPE)
as
BEGIN
INSERT INTO Personnel VALUES (pidPers, pnom, pIndice);
INSERT INTO Chercheur VALUES (pidPers, pspecialite, plaboratoire);
END;
/
```

# Décomposition par distinction

## Contrainte B

- Suppression de données: 2 procédures

– Suppression d'un enseignant

```
CREATE PROCEDURE suppression_enseignant (pIdPers Personne.idPers%TYPE)
as
BEGIN
DELETE FROM Enseignant WHERE idPers= pidPers;
DELETE FROM Personne WHERE idPers= pidPers;
END;
/
```

– Suppression d'un chercheur

```
CREATE PROCEDURE suppression_chercheur (pIdPers Personne.idPers%TYPE)
as
BEGIN
DELETE FROM Chercheur WHERE idPers= pidPers;
DELETE FROM Personne WHERE idPers= pidPers;
END;
/
```

# Décomposition par distinction

## Contrainte B

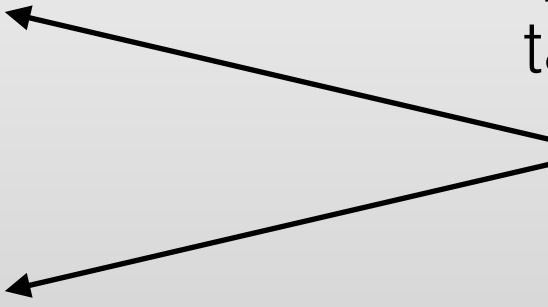
- MAJ de données: 1 trigger

```
CREATE OR REPLACE TRIGGER T_B_U_Personnel
BEFORE UPDATE OF idPers ON Personnel
FOR EACH ROW
BEGIN
    UPDATE Enseignant
    SET idPers = :new.idPers
    WHERE idPers = :old.idPers;

    UPDATE Chercheur
    SET idPers = :new.idPers
    WHERE idPers = :old.idPers;

END;
/
```

comme il ne peut pas  
exister un personnel  
présent dans les 2  
tables, seul 1 des 2  
update sera fait.



# Décomposition par distinction

- **Contrainte de partition (A+B)**

- Contrainte A: il n'existe aucun personnel pouvant être à la fois Enseignant et Chercheur : 2 triggers
- Contrainte B: il n'existe pas de personnel n'étant ni Enseignant ni Chercheur: 4 procédures + 1 trigger

- **Contrainte de totalité (B+C)**

- B a été programmée (il faut cependant revoir les procédures de suppression et d'insertion)
- Contrainte C: un personnel peut être à la fois Enseignant et Chercheur -> revient à ne pas programmer la contrainte A

- **Contrainte d'exclusion (A+D)**

- A a été programmée
- Contrainte D: un personnel peut n'être ni Enseignant ni Chercheur. revient à ne pas programmer la contrainte B

- **Sans contrainte (C+D)**

- Aucune contrainte n'est à programmer