Transformation de l'héritage en relationnel

Table des matières

Objectifs

Prérequis :

- Savoir faire un MCD UML avec des classes, des associations simples, de l'héritage.
- Savoir faire un MLD relationnel à parti d'un MCD UML avec des classes et des associations simples.

Cours

1. Les trois représentations de l'héritage en relationnel

Objectifs

Savoir faire le passage d'un schéma conceptuel UML à un schéma relationnel.

Connaître les choix possibles pour le cas de l'héritage et savoir faire les bons choix.

La traduction en relationnel des héritages modélisés au niveau conceptuel peut se faire de plusieurs façons, et le choix demande d'étudier plus en détail la nature de cet héritage.

1.1. Transformation de la relation d'héritage

Le modèle relationnel ne permet pas de représenter directement une relation d'héritage, puisque que seuls les concepts de relation et de clé existent dans le modèle. Il faut donc appauvrir le modèle conceptuel pour qu'il puisse être représenté selon un schéma relationnel.

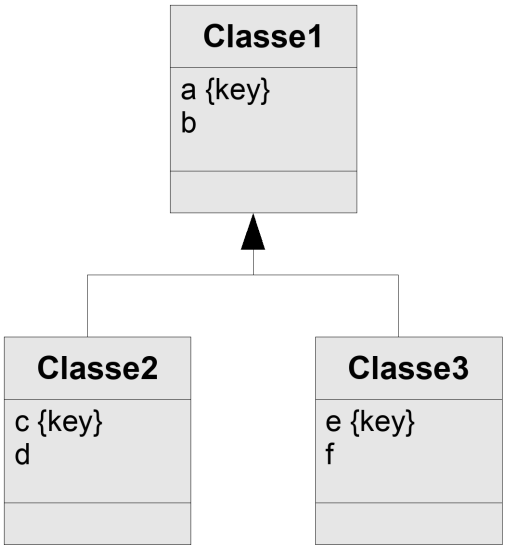
Attention

L'héritage est toujours délicat à traduire en relationnel.

Fondamental

Trois solutions existent pour transformer une relation d'héritage :

- Représenter l'héritage par une référence entre la classe mère et la classe fille.
- Représenter uniquement les classes filles par une relation chacune.
- Représenter uniquement la classe mère par une seule relation.



Héritage

Méthode

En pratique le choix entre ces trois solutions va dépendre de l'étude détaillée de l'héritage :

- L'héritage est-il complet ?
- La classe mère est-elle abstraite ?
- Quelles sont les associations qui lient la classe mère et les classes filles à d'autres classes ?

Rappel

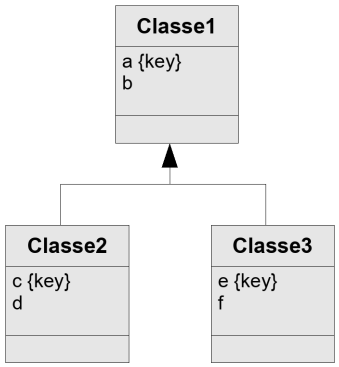
Classe abstraite *(cf. p.20)*

Héritage complet *(cf. p.22)*

1.2. Transformation de la relation d'héritage par référence (des filles vers la mère)

Méthode

- Chaque classe, mère ou fille, est représentée par une relation.
- La clé primaire de la classe mère est utilisée pour identifier chacune de ses classes filles : cette clé étant pour chaque classe fille à la fois la clé primaire et une clé étrangère vers la classe mère.



Héritage

```
Classe1(#a,b)
Classe2(#a=>Classe1,c,d) avec c KEY
Classe3(#a=>Classe1,e,f) avec e KEY
```

Remarque

Si une classe fille a une clé définie dans le modèle conceptuel, cette clé n'est pas retenue pour être la clé primaire dans le modèle relationnel, étant donné que c'est la clé étrangère référence à la classe mère qui est retenue.

Remarque

La cardinalité d'un lien entre une classe fille et une classe mère est (1,1):(0,1) : En effet toute instance fille référence obligatoirement une et une seule instance mère (pas d'héritage multiple) et toute instance mère est référencée une ou zéro fois (zéro fois si un objet peut être directement du type de la classe mère) par chaque instance fille.

Complément

Héritage par une référence et vues *(cf. p.22)*

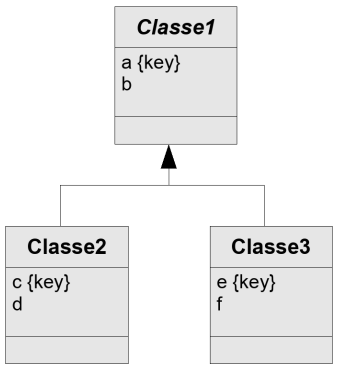
Méthode alternative **Complément**

Transformation de la relation d'héritage par référence (de la mère vers les filles) *(cf. p.25)*

1.3. Transformation de la relation d'héritage par les classes filles

Méthode

- Chaque classe fille est représentée par une relation, la classe mère n'est pas représentée (si elle est abstraite).
- Tous les attributs de la classe mère sont répétés au niveau de chaque classe fille.
- La clé primaire de la classe mère est utilisée pour identifier chacune de ses classes filles.



Héritage (classe mère abstraite)

```
Classe2(#a,b,c,d) avec c KEY
Classe3(#a,b,e,f) avec e KEY
```

Remarque

Si une classe fille a une clé primaire au niveau du MCD, cette clé n'est pas retenue, et c'est bien la clé héritée de la classe mère qui devient la clé primaire (mais elle est bien entendu maintenue comme clé candidate).

Héritage exclusif **Complément**

Cette solution est adaptée dans le cas d'un héritage exclusif, c'est à dire si aucun objet d'une classe fille n'appartient aussi à une autre classe fille. Dans le cas contraire, le problème est que des redondances vont être introduites puisqu'un même tuple devra être répété pour chaque classe fille à laquelle il appartient.

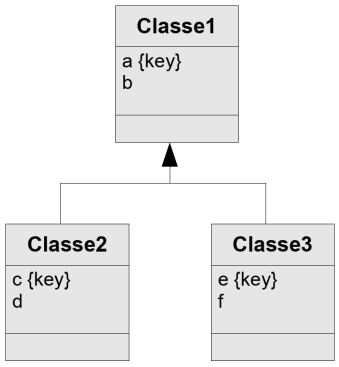
Complément

Héritage par les classes filles et vues *(cf. p.25)*

1.4. Transformation de la relation d'héritage par la classe mère

Méthode

- Seule la classe mère est représentée par une relation (ses classes filles ne sont pas représentées par des relations).
- Tous les attributs de chaque classe fille sont réintégrés au niveau de la classe mère.
- La clé primaire de la classe mère est utilisée pour identifier la relation.
- Un attribut supplémentaire de discrimination `t` (pour "type"), est ajouté à la classe mère, afin de distinguer les tuples. Cet attribut est de type énumération et a pour valeurs possibles les noms de la classe mère ou des différents classes filles.



Héritage

```
Classe1(#a,b,c,d,e,f,t:{1,2,3}) avec c UNIQUE, e UNIQUE, t NOT NULL
```

Remarque

Si une classe fille a une clé primaire propre, cette clé sera réintégrée à la classe mère, au même titre qu'un autre attribut, mais elle n'officiera pas en tant que clé candidate car elle pourra contenir des valeurs nulles (elle sera néanmoins unique).

Héritage complet

Conseil

Cette solution est optimum dans le cas d'un héritage complet, c'est à dire si les classes filles ne définissent pas d'attributs autre que ceux hérités. Dans le cas contraire cela engendre des valeurs `null`.

En pratique l'héritage complet est rare, mais nombre d'héritages le sont presque et pourront alors être raisonnablement gérés selon cette approche.

Héritage non exclusif : représentation de la discrimination par des booléens

Complément

Si l'héritage concerne un nombre élevé de classes filles et qu'il est principalement non exclusif alors le domaine de l'attribut de discrimination peut impliquer une combinatoire importante de valeurs. Pour contourner cette lourdeur il est possible d'utiliser, en remplacement, un attribut de domaine booléen pour chaque classe fille spécifiant si un tuple est un objet de cette classe.

```
Classe1(#a,b,c,d,e,f,t:{1,2,3})
```

équivaut à :

```
Classe1(#a,b,c,d,e,f,t2:boolean:,t3:boolean)
```

Contraintes :

- `avec t2 X t3` si l'héritage est exclusif
- `avec t2 XOR t3` si l'héritage est exclusif et la classe mère abstraite

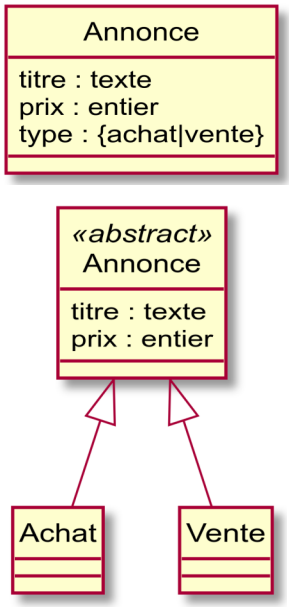
Complément

Héritage par la classe mère et vues *(cf. p.29)*

1.5. Exercice

Exercice

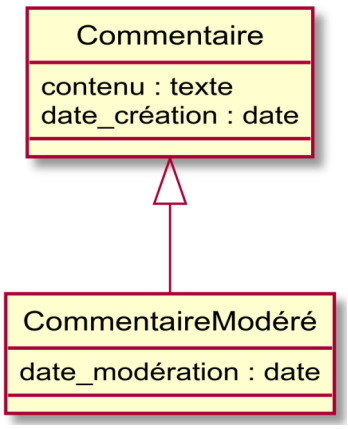
Les deux modèles UML suivants sont-ils équivalents au sens où ils conduiront au même modèle relationnel ?



- ○ Les deux modèles sont équivalents
- ○ Les deux modèles ne sont pas équivalents

Exercice

Les deux modèles UML suivants sont-ils équivalents au sens où ils conduiront au même modèle relationnel ?



- ○ Les deux modèles sont équivalents
- ○ Les deux modèles ne sont pas équivalents

2. Choisir la meilleure modélisation de l'héritage en relationnel

2.1. Choix de la transformation

Les trois solutions de transformation fonctionnent dans tous les cas, mais avec plus ou moins de simplicité (en particulier du point de vue de l'implémentation en SQL). La difficulté consiste pour chaque relation d'héritage à choisir le bon mode de transformation, sachant que chaque solution possède ses avantages et ses inconvénients.

Fondamental

L'objectif est de choisir la transformation qui conduit au modèle relationnel le plus simple à implémenter.

Méthode générale *Méthode*

La méthode générale consiste à réaliser les trois transformations pour choisir la plus simple. Avec l'expérience on peut écarter rapidement certaines solutions typiques, mais dès que l'on sort de ces cas, il est conseillé de procéder de façon systématique.

Inconvénients typiques de chaque méthode *Méthode*

- Par référence
  - Lourdeur liée à la nécessité de représenter les données des classes filles sur deux relations.
  - Contrainte complexe systématique (dans le cas de l'héritage exclusif)
  - Contrainte complexe lorsque la classe mère est abstraite
- Par les classes filles
  - Contraintes complexes lorsque la classe mère n'est pas abstraite ou qu'elle possède des attributs uniques
  - Contraintes complexes dans le cas de certaines associations avec la classe mère
- Par la classe mère
  - Présence de valeurs nulles systématiques dans la cas de l'héritage non complet
  - Contraintes simples lorsque l'héritage est presque complet (réalisables avec des CHECK en SQL)
  - Contraintes complexes lorsque l'héritage n'est pas complet

Complément

Typologie des contraintes en bases de données (cf. p.31)

Penser aux vues ! *Complément*

Il est important de bien penser à ajouter les vues permettant de retrouver le schéma initialement recherché.

Héritage par une référence et vues (cf. p.22)

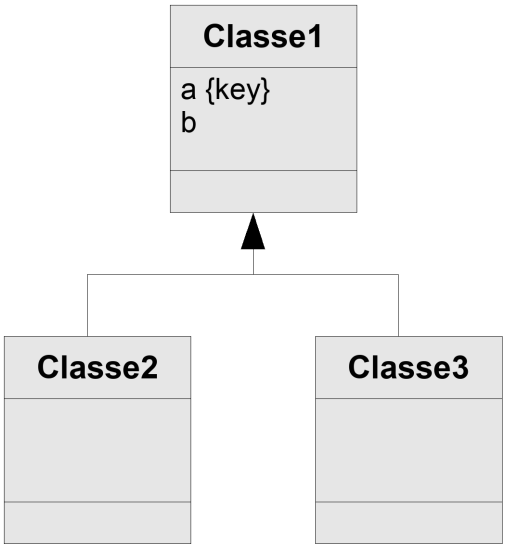
Héritage par les classes filles et vues (cf. p.25)

Héritage par la classe mère et vues (cf. p.29)

2.2. Cas simples de transformation par la classe mère

Héritage complet *Méthode*

Dans ce cas, choisir un **héritage par la classe mère**.

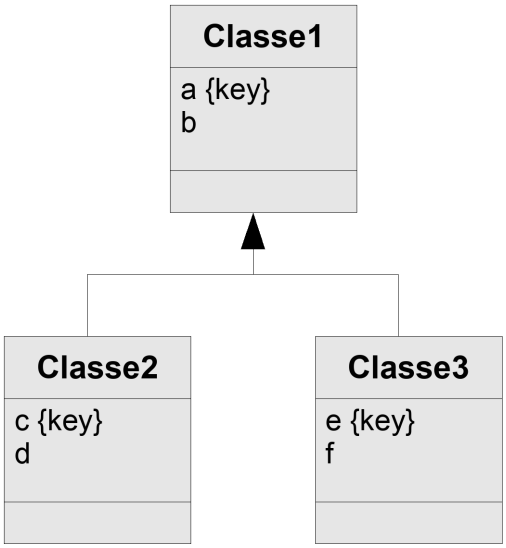


Héritage complet

```
Classe1(#a,b,t:{1,2,3})
```

Héritage presque complet *Méthode*

L'héritage presque complet peut être géré comme l'héritage complet, **par la classe mère**, surtout si les classes filles ne possèdent pas de clé propre.



Héritage presque complet (classe mère non abstraite)

`Classe1(#a,b,c,d,e,f,t:{1,2,3}) avec c UNIQUE et e UNIQUE`

Contraintes : on devra vérifier que les nullités et non nullités de c, d, e et f en fonction du type t (cela peut se faire grâce à un CHECK en SQL)

Complément

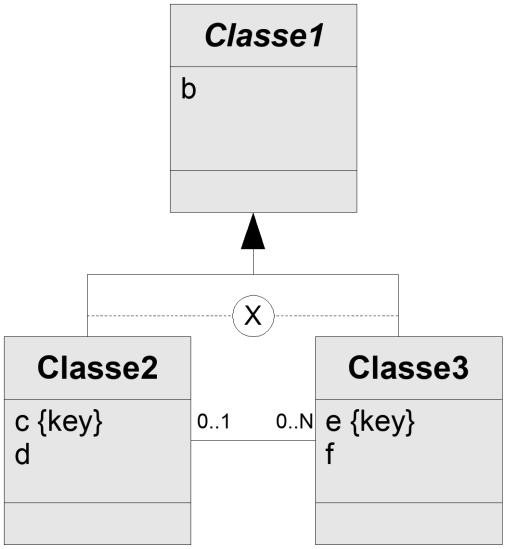
Contraintes de l'héritage par la classe mère *(cf. p.31)*

2.3. Cas simples de transformation par les classes filles

Héritage non complet avec classe mère abstraite (sans association ni clé au niveau de la classe mère)

Méthode

Dans ce cas, choisir un **héritage par les classes filles**.



Héritage exclusif non complet (classe mère abstraite)

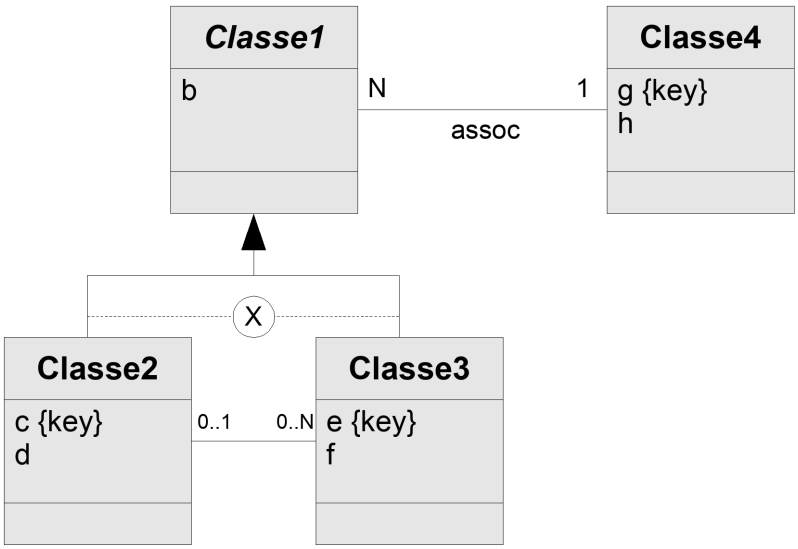
`Classe2(#c,b,d)`

`Classe3(#e,b,f,fka=>Classe2)`

Héritage non complet avec classe mère abstraite (sans association ni clé au niveau de la classe mère)

Méthode

Si la classe mère possède une association sortante (c'est elle qui référence, mais elle n'est jamais référencée), alors l'**héritage pas les classes filles** fonctionne toujours très bien.



Héritage exclusif non complet (classe mère abstraite, association sortante)

```
Classe2(#c,b,d, fkg=>Classe4)
Classe3(#e,b,f,fka=>Classe2, fkg=>Classe4)
Classe4(#g,h)
```

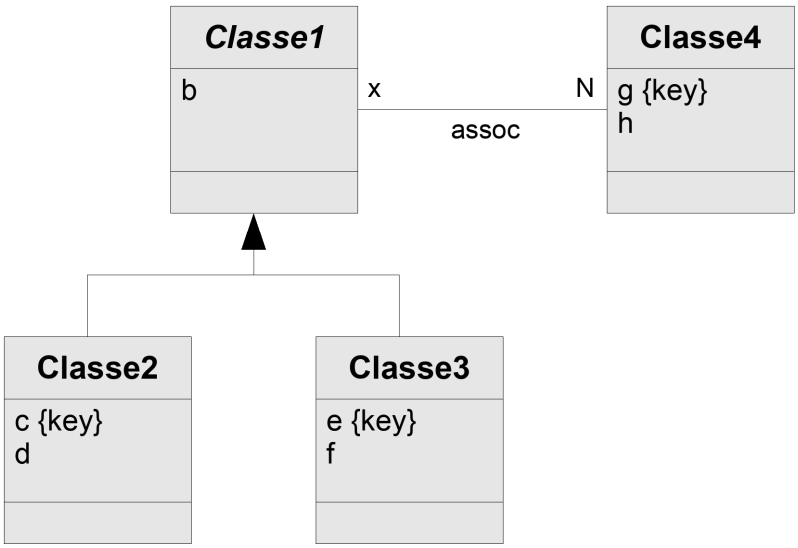
Méthode

Contraintes de l'héritage par les classes filles (cf. p.33)

2.4. Cas problématiques

Héritage par les classes filles avec association M:N ou 1:N (entrante) sur la classe mère

Attention



Héritage avec association x:N sur la classe mère

On traite le cas x=1 (1:N), mais le cas M:N ne change rien au problème.

```
Classe2(#c,b,d)
```

`Classe3(#e,b,f)`

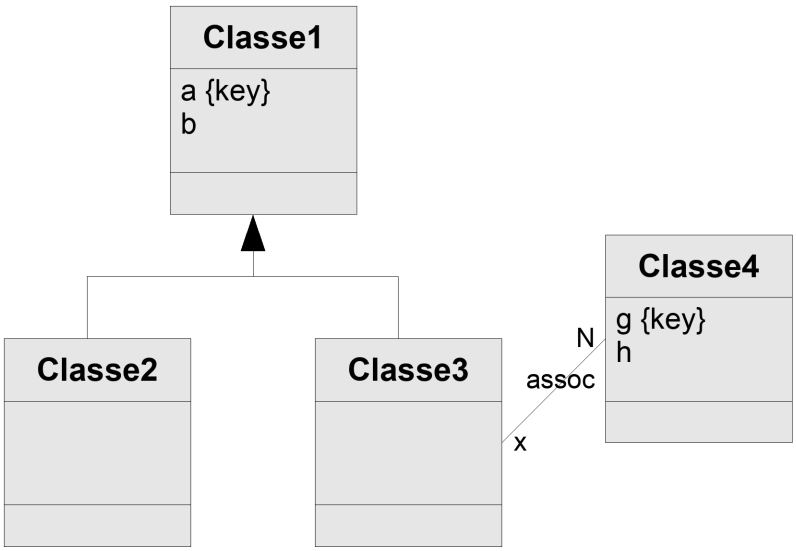
`Classe4(#g,h,fka=>Classe2, fkb=>Classe3)`

Contrainte : `fka OR fkb`

La solution consiste à ajouter autant de clés étrangères que de classes filles et à gérer le fait que ces clés ne peuvent pas être co-valuées.

Héritage non complet par la classe mère (association sur une classe fille)

Attention



Héritage non complet

On traite le cas x=1 (1:N).

`Classe1(#a,b,t:{1,2,3})`

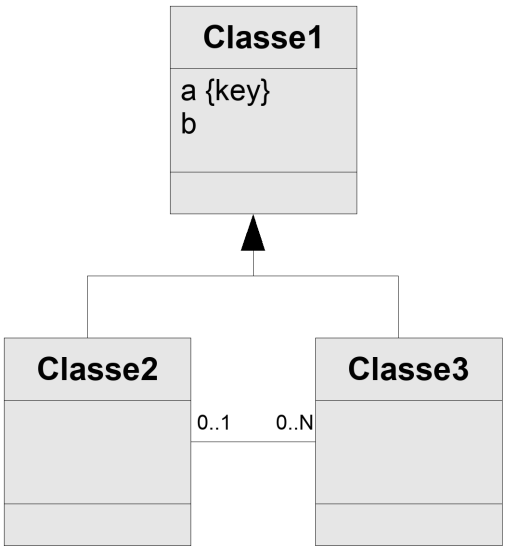
`Classe4(#g,h,fka=>Classe1)`

Contraintes : `Classe4.fka ne référence que des enregistrements tels que Classe1.t=3`

On est obligé d'ajouter une contrainte pour limiter la portée de la clé étrangère de Classe4 ; on est sorti ici de ce que l'on sait faire de façon simple en relationnel.

Héritage non complet par la classe mère (association entre classes filles)

Attention



Héritage non complet

`Classe1(#a,b,fka=>Classe1,t:{1,2,3})`

Contraintes : `fka ne référence que des enregistrements tels que t=2 ; si fka alors t=3`

Dans ce cas la solution est encore plus problématique, elle permettra en l'état des associations entre Classe1 et Classe3, et même entre Classe3 et Classe3, on est très loin de la modélisation conceptuelle initiale.

Conseil

Afin de déterminer si un héritage est presque complet ou non, il faut donc surtout regarder les associations, ce sont elles qui poseront le plus de problème un fois en relationnel (à cause de l'intégrité référentielle).

Héritage non exclusif *Complément*

L'héritage non exclusif ne doit pas être traité par les classes filles, sous peine d'introduire de la redondance.

Héritage multiple *Complément*

L'héritage multiple sera généralement mieux géré avec un héritage par référence.

Complément

Héritage complet *(cf. p.22)*

Classes abstraites *(cf. p.20)*

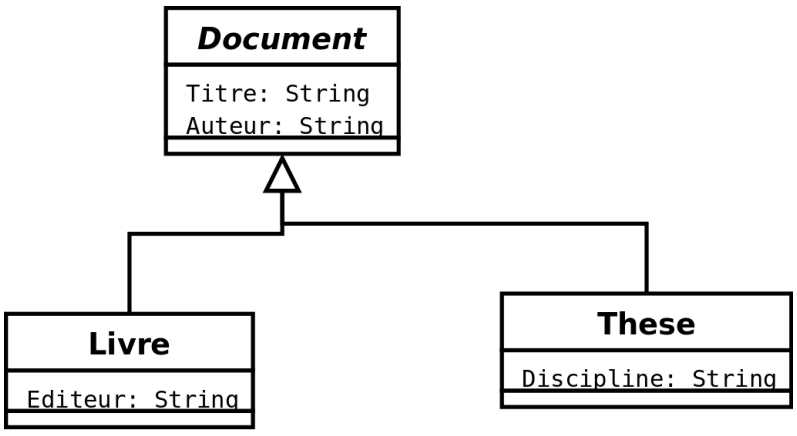
Les cas problématiques obligent à ajouter des contraintes *Complément*

Contraintes en UML *(cf. p.34)*

Liste des contraintes en R *(cf. p.35)*

2.5. Exemple de transformation d'une relation d'héritage

Exemple



Représentation de documents

Type d'héritage *Remarque*

1. L'héritage n'est pas complet, puisque l'on observe que les thèses et les livres ont des attributs qui ne sont pas communs (la discipline pour la thèse et l'éditeur pour le livre). Mais l'on peut considérer qu'il est **presque complet**, car il n'y a qu'un attribut par classe fille de différence et pas d'association associée aux classes filles.
2. La classe mère est **abstraite**, on ne gère que des livre ou des thèses, pas d'autres documents ; il n'y a aucune clé candidate identifiée.

La meilleure solution est ici un **héritage par les classes filles** ; un héritage par la classe mère est le second choix (contraintes simples) ; en revanche l'héritage par référence est un mauvais choix (contraintes complexes).

Observons les avantages et inconvénients de chacune des trois solutions.

Héritage absorbé par les classes filles

```
These(#id:int, titre:text, discipline:text, auteur:text)

Livre(#id:int, titre:text, editeur:text, auteur:text)
```

Héritage représenté par une référence

```
Document(#id:int, titre:text, auteur:text) avec titre et auteur NOT NULL

These(#id=>Document, discipline:text)

Livre(#id=>Document, editeur:text)

Contraites compliquées :
- Intersection (PROJECTION(These, id), Projection(Livre, id)) = {}
- Projection(Document,id) = Union (Projection(These,id), Projection(Livre,id))

Vues :
vThese = JointureNaturelle (Document, These)
vLivre = JointureNaturelle (Document, Livre)
```

Héritage absorbé par la classe mère

```
Document(#id:int, titre:text, discipline:text, editeur:text, auteur:text, type:
{These|Livre})

Contraintes simples :
- (discipline NOT NULL AND type=These) OR (editeur NOT NULL AND type=Livre)
- NOT (discipline NOT NULL AND type=Livre)
- NOT (editeur NOT NULL AND type=These)

Vues :
vThese = Projection (Restriction (Document, type=These), id, titre, discipline,
auteur)
vLivre = Projection (Restriction (Document, type=Livre), id, titre, editeur,
auteur)
```

Exercices

1. Exercice : Lab II+

[45 min]

Description du problème

Un laboratoire souhaite gérer les médicaments qu'il conçoit.

- Un médicament est décrit par un nom, qui permet de l'identifier. En effet il n'existe pas deux médicaments avec le même nom. Un médicament comporte une description courte en français, ainsi qu'une description longue en latin. On gère aussi le conditionnement du médicament, c'est à dire le nombre de pilules par boîte (qui est un nombre entier).
- À chaque médicament on associe une liste de contre-indications, généralement plusieurs, parfois aucune. Une contre-indication comporte un code unique qui l'identifie, ainsi qu'une description. Une contre-indication est toujours associée à un et un seul médicament.
- Tout médicament possède au moins un composant, souvent plusieurs. Un composant est identifié par un code unique et possède un intitulé. Tout composant peut intervenir dans la fabrication de plusieurs médicaments. Il existe des composants qui ne sont pas utilisés pour fabriquer des médicaments et que l'on veut quand même gérer.
- Il existe des composants naturels et des composants artificiels. Pour les composants naturels, on gère l'espèce végétale qui porte le composant. Pour les composants artificiels, on gère le nom de la société qui le fabrique.

Exemple de données

Afin de matérialiser notre base de données, nous obtenons les descriptions suivantes :

- Le **Chourix** a pour description courte *« Médicament contre la chute des choux »* et pour description longue *« Vivamus fermentum semper porta. Nunc diam velit, adipiscing ut tristique vitae, sagittis vel odio. Maecenas convallis ullamcorper ultricies. Curabitur ornare. »*. Il est conditionné en boîte de 13.

  Ses contre-indications sont :

  - CI1 : Ne jamais prendre après minuit.
  - CI2 : Ne jamais mettre en contact avec de l'eau.

  Ses composants sont le **HG79** et le **SN50**.

- Le **Tropas** a pour description courte *« Médicament contre les dysfonctionnements intellectuels »* et pour description longue *« Suspendisse lectus leo, consectetur in tempor sit amet, placerat quis neque. Etiam luctus porttitor lorem, sed suscipit est rutrum non. »*. Il est conditionné en boîte de 42.

  Ses contre-indications sont :

  - CI3 : Garder à l'abri de la lumière du soleil

  Son unique composant est le **HG79**.

- Les composants existants sont :
    - **HG79** : "Vif-argent allégé" ; il s'agit d'un composant naturel extrait de l'**edelweiss**.
    - **HG81** : "Vif-argent alourdi" ; il s'agit aussi d'un composant naturel extrait de l'**edelweiss**.
    - **SN50** : "Pur étain" ; il s'agit d'un composant artificiel fabriqué par **Lavoisier et fils SA**.

Question 1

Réaliser le modèle conceptuel de données en UML du problème.

Question 2

En mobilisant les règles adéquates, proposer un modèle logique de données correspondant en relationnel. Le repérage des domaines et des clés est obligatoire.

Question 3

Dessiner des tableaux remplis avec les données fournies en exemple, afin de montrer que le modèle fonctionne selon le besoin exprimé initialement. On pourra mettre le premier mot seulement des descriptions pour gagner du temps.

2. Exercice : Parc informatique

[30 minutes]

Vous avez en charge la réalisation d'un modèle de base de données pour la gestion d'un parc informatique.

L'analyse des besoins révèlent les informations suivantes : tout matériel informatique est identifié de façon unique par un numéro de série et est décrit par une désignation. Il existe trois types de matériel informatique distincts : les PC, les serveurs et les imprimantes. Pour les PC, les informations que l'on veut gérer sont la taille de la mémoire vive et la cadence du micro-processeur, pour les serveurs on veut gérer leur volume de disque dur et pour les imprimantes leur résolution maximale d'impression.

On veut également gérer les connexions au réseau sachant que tout PC peut être relié à un ou plusieurs serveurs et que chaque serveur sert bien entendu plusieurs PC ; et qu'un PC peut être relié à une imprimante, qui est également utilisée par plusieurs PC.

Question 1

Réaliser le modèle conceptuel UML de ce problème.

Question 2

Réalisez le passage au modèle logique relationnel.

3. Exercice : Literie

[20 min]

Un revendeur de matelas et de sommier a besoin de créer une base de données, afin de générer son catalogue des produits. Pour cela, il fait appel à vous, afin de concevoir une base de données qui puisse prendre en compte ses besoins. Voici des extraits du document qui retrace les besoins, rédigé par le client :

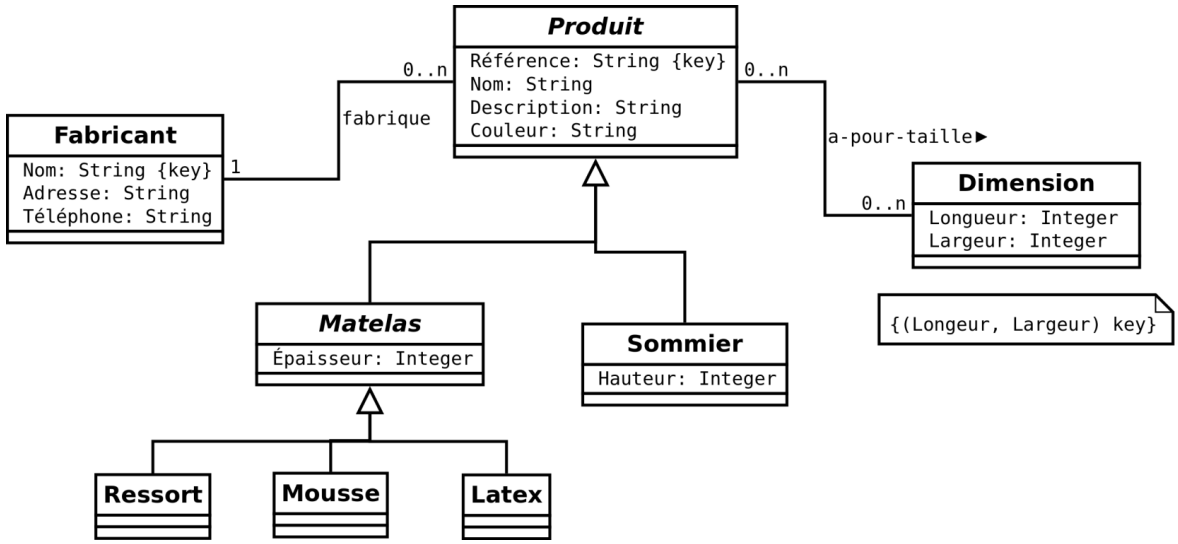
« Le fonctionnement actuel est relativement simple, nous souhaitons juste l'automatiser pour éviter les erreurs et les pertes de temps. Nous avons des types de produits, qui sont des matelas ou des sommiers. Chaque type de produit est caractérisé par des références uniques que nous créons en plus du nom posé par le fabricant (ce nom reprend parfois un plus ancien d'un concurrent) ; nous ajoutons souvent une description textuelle et précisons toujours la couleur. »

« Un type de produit est toujours lié à un fabricant, caractérisé par son nom (sa marque), et nous avons toujours au minimum une adresse et un numéro de téléphone. Un type de matelas a en plus une épaisseur, nécessaire pour l'ajouter dans les catalogues et les publicités. Un matelas est aussi catégorisé en fonction du matériau utilisé : ressort, latex ou mousse. Un type de sommier possède toujours une hauteur. »

« Chaque type de produit est proposé en une ou plusieurs dimensions (longueur et largeur, qui sont souvent des dimensions standardisées). »

Diagramme UML

Un stagiaire a déjà proposé le diagramme UML ci-après.



Literie

Question 1

Vérifiez que le diagramme est correct par rapport à l'énoncé.

Question 2

Proposez un modèle relationnel à partir de l'UML de la question précédente

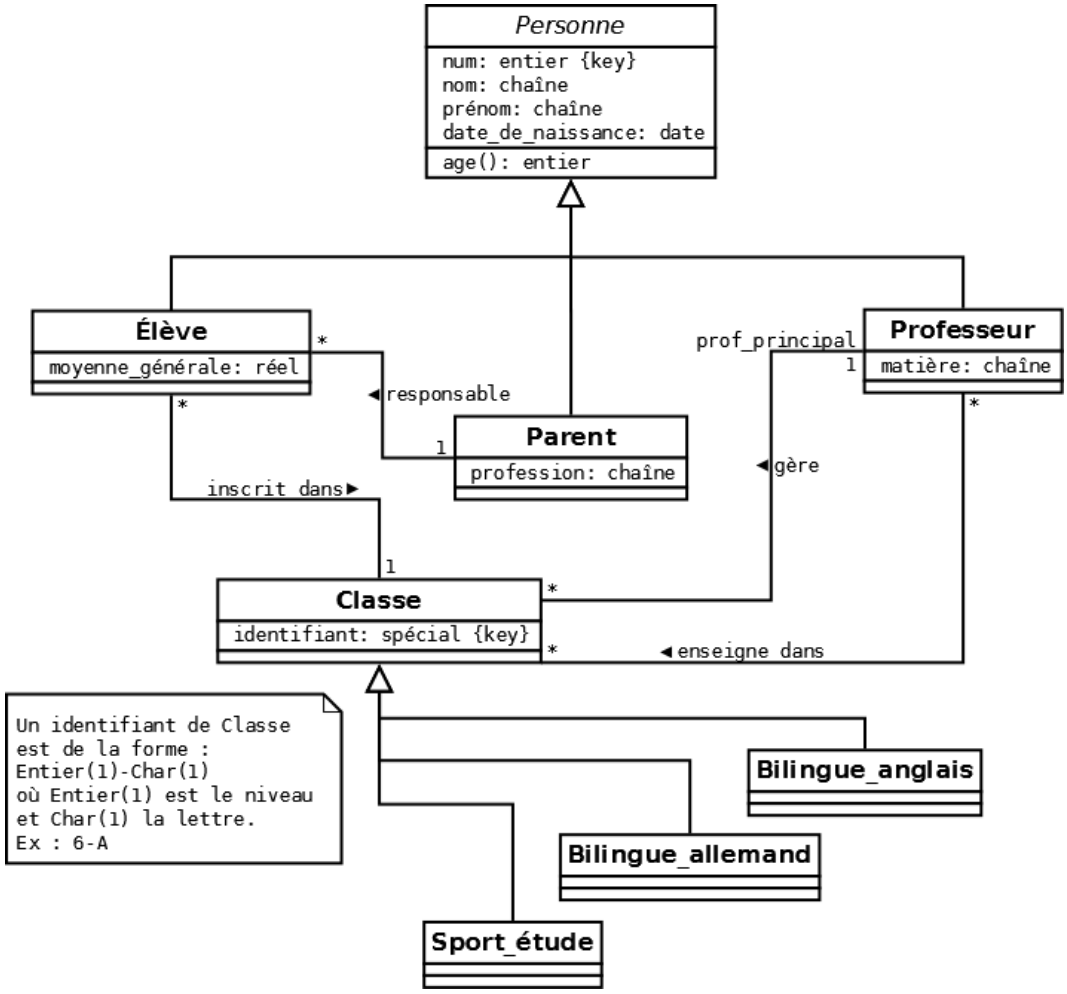
4. Exercice : À l'école

[20 min]

Une école souhaite se doter d'une base de données pour suivre ses effectifs (élèves et professeurs) ainsi que les classes auxquelles ils sont rattachés. L'analyse des besoins est la suivante :

- Les classes sont identifiées par un niveau et une lettre (6-A, 6-B, 5-A, ...).
- Les classes peuvent avoir un programme adapté suivant un thème unique (sport-étude, bilingue anglais ou bilingue allemand).
- Les classes ont un professeur principal et plusieurs professeurs intervenant dans leur spécialité.
- Les classes accueillent des élèves.
- Les élèves sont sous la responsabilité d'un parent (qui peut aussi être professeur, voire lui même élève).

On a élaboré le diagramme UML ci-après.



École

Question

Proposer un schéma relationnel. Justifier les transformation d'héritage.

Contenus annexes

1. Classe abstraite

Classe abstraite *Définition*

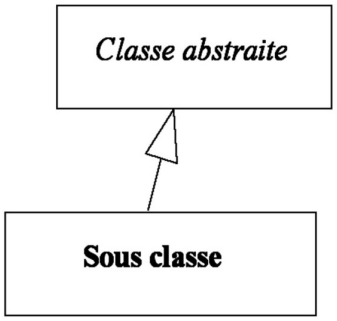
Une classe abstraite est une classe non instanciable. Elle exprime donc une généralisation abstraite, qui ne correspond à aucun objet existant du monde.

Classe abstraite et héritage *Attention*

Une classe abstraite est **toujours héritée**. En effet sa fonction étant de généraliser, elle n'a de sens que si des classes en héritent. Une classe abstraite peut être héritée par d'autres classes abstraites, mais en fin de chaîne des classes non abstraites doivent être présentes pour que la généralisation ait un sens.

Italique *Syntaxe*

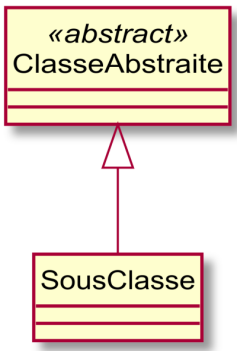
On note les classes abstraites en italique.



Notation d'une classe abstraite en UML

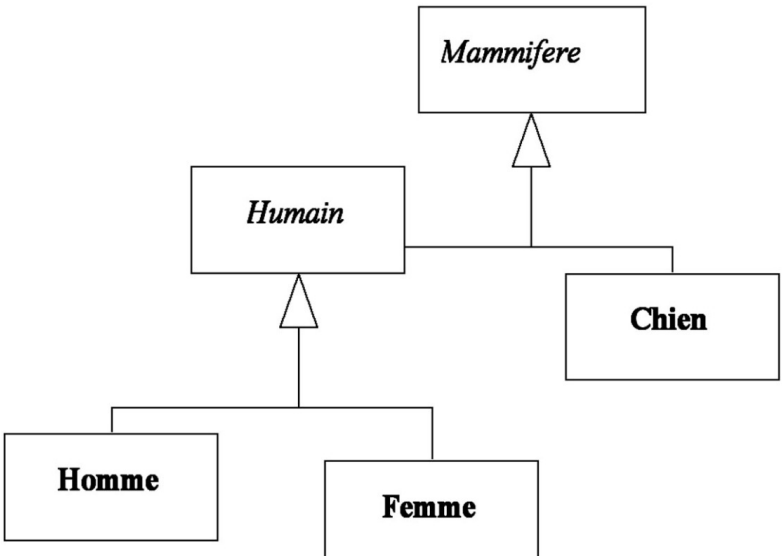
<<abstract>> *Syntaxe*

On peut utiliser le stéréotype `<<abstract>>` pour noter les classes abstraites (cette représentation est plus évidente à lire).



Notation <<abstract>>

Exemple de classes abstraites *Exemple*



Des chiens et des hommes

Dans la représentation précédente on a posé que les hommes, les femmes et les chiens étaient des objets instanciables, généralisés respectivement par les classes mammifère et humain, et mammifère.

Selon cette représentation, il ne peut donc exister de mammifères qui ne soient ni des hommes, ni des femmes ni des chiens, ni d'humains qui ne soient ni des hommes ni des femmes.

Complément

Stéréotype *(cf. p.21)*

2. Stéréotype

Stéréotype UML *Définition*

Un stéréotype UML est une syntaxe permettant d'ajouter de la sémantique à la modélisation des classes. Il permet de définir des **types de classe**, afin de regrouper conceptuellement un ensemble de classes (à l'instar d'une classe qui permet de regrouper conceptuellement un ensemble d'objets).

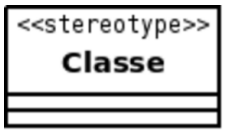
C'est une mécanique de méta-modélisation : elle permet d'étendre le méta-modèle UML, c'est à dire le modèle conceptuel du modèle conceptuel.

Méta-modèle *Définition*

Un méta-modèle est le modèle d'un modèle. Par exemple le méta-modèle UML comprend les concepts de classe, attribut, association, cardinalité, composition, agrégation, contraintes, annotations, ... On mobilise ces concepts (on les instancie) pour exprimer un modèle particulier suivant le formalisme UML.

Les stéréotypes permettent donc d'ajouter au méta-modèle UML standard, celui que tout le monde utilise, des concepts locaux pour enrichir le langage de modélisation que l'on utilise pour réaliser des modèles.

Syntaxe



Notation d'un stéréotype en UML

Stéréotypes spécifiques et stéréotypes standard *Conseil*

Un stéréotype spécifique enrichit le méta-modèle UML, mais selon une sémantique qui est propre à celui qui l'a posé, non standard donc. La conséquence est que pour un tiers, l'interprétation du stéréotype n'est plus normalisée, et sera potentiellement plus facilement erronée. Il convient donc de ne pas abuser de cette mécanique.

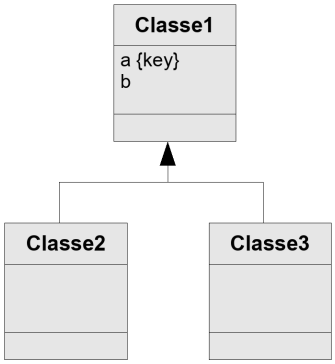
Deux ou trois stéréotypes spécifiques, correctement définis, sont faciles à transmettre, plusieurs dizaines représenteraient un nouveau langage complet à apprendre pour le lecteur du modèle.

Il existe des stéréotypes fournit en standard par UML, ou communément utilisés par les modélisateurs. L'avantage est qu'il seront compris plus largement, au même titre que le reste du méta-modèle (ils ont une valeur de standard).

3. Héritage complet

Héritage complet et presque complet *Définition*

Un héritage est complet si ses classes filles n'ont aucune caractéristiques (attributs, méthodes, associations) propres.



Héritage complet

Un héritage est presque complet si les classes filles ont des méthodes propres, quelques attributs propres, et **aucune association propre**.

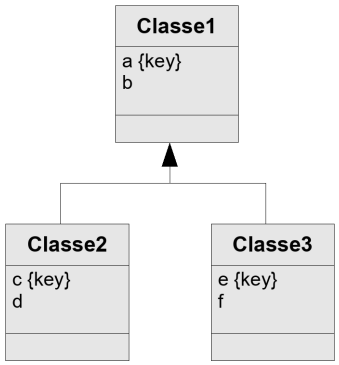
4. Héritage par une référence et vues

Rappel

Transformation de la relation d'héritage par référence *(cf. p.5)*

Méthode

Une vue est créée pour chaque classe fille en réalisant une jointure avec la classe mère.



Héritage

```
Classe1(#a,b)
Classe2(#a=>Classe1,c,d) avec c KEY
Classe3(#a=>Classe1,e,f) avec e KEY
vClasse2=jointure(Classe1,Classe2,a=a)
vClasse3=jointure(Classe1,Classe3,a=a)
```

Exemple

Soit la classe A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A, comprenant la clé K' et les attributs B1 et B2.

Le modèle relationnel correspondant selon cette transformation est :

```
1 A (#K, A1, A2)
2 B (#K=>A, K', B1, B2)
3 vB = Jointure (A, B, A.K=B.K)
```

Algèbre relationnel

Complément

Jointure *(cf. p.23)*

5. Jointure

Jointure

Définition

La jointure est une opération binaire (c'est à dire portant sur deux relations). La jointure de R1 et R2, étant donné une condition C portant sur des attributs de R1 et de R2, **de même domaine**, produit une relation R3 ayant pour schéma la juxtaposition de ceux des relations R1 et R2 et pour tuples l'ensemble de ceux obtenus par concaténation des tuples de R1 et de R2, et qui vérifient la condition C.

Syntaxe

```
R = Jointure (R1, R2, condition)
```

Exemple

Soit les deux relations suivantes : `Personne (nom, prénom, age)` et `Voiture (type, marque, propriétaire)`

Personne

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Voiture

type	marque	propriétaire
Tesla	Model X	Dupont
Citroën	2 CV	Durand
Citroën	3 CV	Dupont

Soit l'opération suivante : `R = Jointure (Personne, Voiture, Personne.Nom=Voiture.Propriétaire)`

On obtient alors la relation R composée des tuples suivants :

R

nom	prénom	age	type	marque	propriétaire
Dupont	Pierre	20	Tesla	Model X	Dupont
Dupont	Pierre	20	Citroën	3 CV	Dupont
Durand	Jean	30	Citroën	2 CV	Durand

Fondamental

La jointure est l'opération qui permet de rassembler les informations séparées entre plusieurs tables et référencées par des clés étrangères.

Opération additionnelle

Remarque

La jointure n'est pas une opération de base, elle peut être réécrite en combinant le produit et la restriction.

Équi-jointure

Complément

Une équi-jointure est une jointure dont la condition est un test d'égalité.

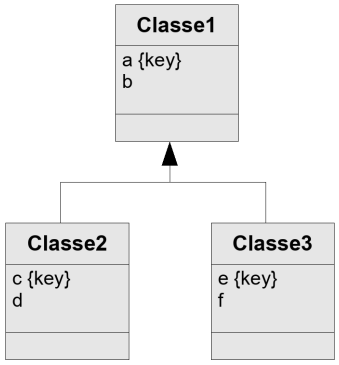
Syntaxes alternatives *Complément*

```
R = ⋈ (R1, R2, condition)
R = R1 ⋈(condition) R2
```

6. Transformation de la relation d'héritage par référence (de la mère vers les filles)

Méthode

- Chaque classe, mère ou fille, est représentée par une relation.
- La classe mère référence chacune de ses classes filles



Héritage

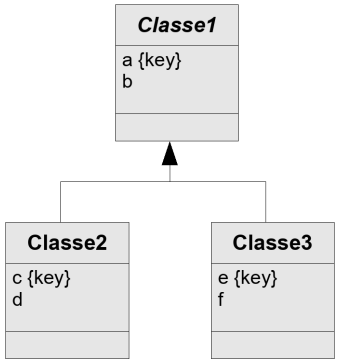
```
Classe1(#a,b,c=>Classe2, e=>Classe3) avec c UNIQUE et e UNIQUE
Classe2(#c,d)
Classe3(#e,f)
```

7. Héritage par les classes filles et vues

Rappel

Transformation de la relation d'héritage par les classes filles *(cf. p.6)*

Héritage absorbé par les classes filles (classe mère abstraite) *Méthode*

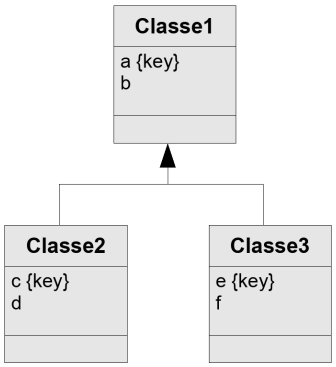


Héritage (classe mère abstraite)

```
Classe2(#a,b,c,d) avec c KEY
Classe3(#a,b,e,f) avec e KEY
```

```
vClasse1=Union(Projection(Classe2,a,b),Projection(Classe3,a,b))
```

Héritage absorbé par les classes filles (classe mère non abstraite) *Méthode*



Héritage

```
Classe1(#a,b)
Classe2(#a,b,c,d) avec c KEY
Classe3(#a,b,e,f) avec e KEY
vClasse1=Union(Union(Classe1,Projection(Classe2,a,b)),Projection(Classe3,a,b))
```

Héritage absorbé par les classes filles *Exemple*

Soit la classe abstraite A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A avec les attributs B1 et B2. Soit la classe C, classe fille de A avec les attributs C1 et C2.

Le modèle relationnel correspondant selon cette transformation est :

```
1 B (#K, A1, A2, B1, B2)
2 C (#K, A1, A2, C1, C2)
3 vA = Union (Projection (B, K, A1, A2), Projection (C, K, A1, A2))
```

Algèbre relationnel *Complément*

Opérateurs ensemblistes *(cf. p.26)*

Projection *(cf. p.28)*

8. Opérateurs ensemblistes

Attention

Les opérateurs ensemblistes sont des relations binaires (c'est à dire entre deux relations) portant sur des relations **de même schéma**.

Union *Définition*

L'union de deux relations R1 et R2 de même schéma produit une relation R3 de même schéma constituée de l'ensemble des tuples appartenant à R1 et/ou à R2.

Différence *Définition*

La différence entre deux relations R1 et R2 de même schéma produit une relation R3 de même schéma constituée de l'ensemble des tuples de R1 n'appartenant pas à R2. Notons que la différence entre R1 et R2 n'est pas égale à la différence entre R2 et R1.

Intersection *Définition*

L'intersection de deux relations R1 et R2 de même schéma produit une relation R3 de même schéma constituée de l'ensemble des tuples appartenant à la fois à R1 et à R2. Notons que l'intersection n'est pas une opération de base, car elle est équivalent à deux opérations de différence successives.

Syntaxe

```
R = Union (R1, R2)
R = Différence (R1, R2)
R = Intersection (R1, R2)
```

Attention

Les opérateurs ensemblistes éliminent les doublons.

Exemple

Soit les deux relations suivantes : `Homme (nom, prénom, age)` et `Femme (nom, prénom, age)`

Homme

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Femme

nom	prénom	age
Martin	Isabelle	24
Blanc	Hélène	25

Soit l'opération suivante : `R = Union (Homme, Femme)`

On obtient alors la relation R composée des tuples suivants :

R

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30
Martin	Isabelle	24
Blanc	Hélène	25

La différence entre Homme et Femme (respectivement entre Femme et Homme) renvoie la relation Homme (respectivement Femme), car aucun tuple n'est commun aux deux relations. L'intersection entre Homme est Femme est vide, pour la même raison.

Union externe *Complément*

Il est possible de définir une opération d'union externe, qui permet de réaliser l'union de deux relations de schéma différent, en ramenant les relations aux mêmes schémas, et en les complétant avec des valeurs nulles.

Syntaxes alternatives *Complément*

```
R = R1 ∪ R2
R = R1 ∩ R1
R = R1 − R2
```

9. Projection

Projection *Définition*

La projection est une opération unaire (c'est à dire portant sur une seule relation). La projection de R1 sur une partie de ses attributs {A1, A2, ...} produit une relation R2 dont le schéma est restreint aux attributs mentionnés en opérande, comportant les mêmes tuples que R1, et dont les doublons sont éliminés.

Syntaxe

```
R = Projection (R1, a1, a2, ...)
```

Exemple

Soit la relation suivante : `Personne (nom, prénom, age)`

Personne

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Soit l'opération : `R = Projection (Personne, nom, age)`

On obtient alors la relation R composée des tuples suivants :

R

nom	age
Dupont	20
Durand	30

La projection élimine les doublons ***Remarque***

Après suppression d'une partie des attributs du schéma, la relation peut comporter des doublons. Étant donné que l'on ne pourrait plus identifier ces doublons les uns par rapport aux autres, la seule solution sensée est donc de considérer que deux doublons sont équivalents, et donc de n'en garder qu'un seul dans la relation résultante.

Syntaxes alternatives ***Complément***

```
R = π (R1, a1, a2, ...)
R = π a1,a2,...(R1)
```

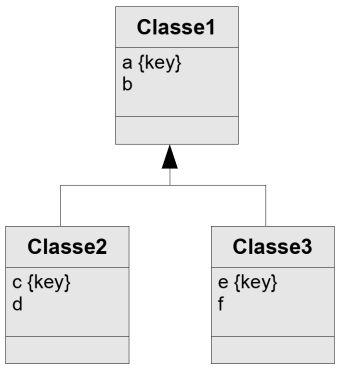
10. Héritage par la classe mère et vues

Rappel

Transformation de la relation d'héritage par la classe mère *(cf. p.7)*

Méthode

Chaque classe est représentée par une vue qui restreint aux tuples de la relation correspondants et les projette sur les attributs correspondants.



Héritage

```
Classe1(#a,b,c,d,e,f,t:{1,2,3}) avec c UNIQUE et e UNIQUE
vClasse1=projection(restriction(Classe1,t=1),a,b)
vClasse2=projection(restriction(Classe1,t=2),a,b,c,d)
vClasse3=projection(restriction(Classe1,t=3),a,b,e,f)
```

Héritage absorbé par la classe mère — *Exemple*

Soit la classe A abstraite avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A avec les attributs B1 et B2. Soit la classe C, classe fille de A avec les attributs C1 et C.

Le modèle relationnel correspondant selon cette transformation est :

```
A (#K, A1, A2, B1, B2, C1, C2, T:{'B','C'})
vB = Projection (Restriction (A, T='B'), K, A1, A2, B1, B2)
vC = Projection (Restriction (A, T='C'), K, A1, A2, C1, C2)
```

Algèbre relationnel — *Complément*

Projection *(cf. p.28)*

Restriction *(cf. p.30)*

11. Restriction

Restriction — *Définition*

La restriction est une opération unaire (c'est à dire portant sur une seule relation). La restriction de R1, étant donnée une condition C, produit une relation R2 de même schéma que R1 et dont les tuples sont les tuples de R1 vérifiant la condition C.

Syntaxe

```
R = Restriction (R1, condition)
```

Exemple

Soit la relation suivante : `Personne (nom, prénom, age)`

Personne

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Soit l'opération suivante : `R = Restriction (Personne, age>25)`

On obtient alors la relation R composée de l'unique tuple restant suivant :

R

nom	prénom	age
Durand	Jean	30

Syntaxes alternatives *Complément*

```
R = σ (R1, condition)
```

$R = \sigma_{condition}(R1)$

Sélection *Complément*

On utilise parfois sélection comme synonyme de restriction, mais il vaut mieux ne pas utiliser ce terme qui prend un sens différent en SQL.

12. Contraintes simples et contraintes compliquées

Les contraintes exprimées dans le cadre de la conception de bases de données peuvent être catégorisées en trois types, du plus simple au plus complexe.

Fondamental

Il faut préférer les contraintes plus simples à celles plus compliquées dans la mesure où elles permettent d'obtenir le modèle recherché (principe du rasoir d'Ockham).

Cela minimisera le travail d'implémentation et de maintenance, et augmentera la fiabilité.

Les contraintes exprimables directement en relationnel

Ce sont les plus simples :

- UNIQUE
- NOT NULL

Les contraintes facilement réalisables en SQL (qui ne concernent qu'une seule relation)

Elles peuvent être exprimées directement en SQL au sein du CREATE TABLE via un CHECK.

Les contraintes compliquées qui nécessitent une couche de programmation en plus du SQL

- On favorisera l'expression de vues (permettant de détecter les anomalies typiquement).
- On utilisera les déclencheurs ou la couche applicative.

13. Contraintes de l'héritage par la classe mère

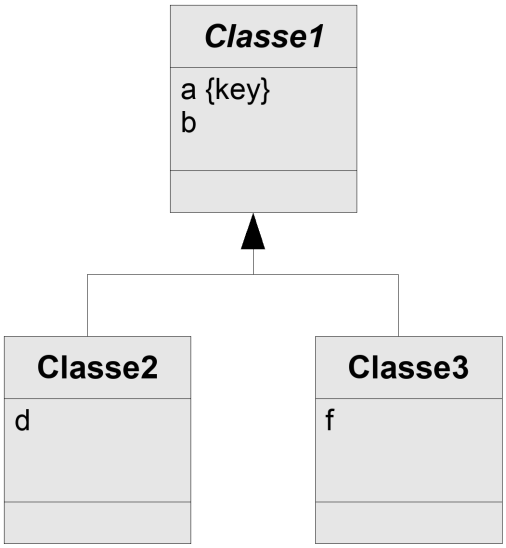
Rappel

Transformation de la relation d'héritage par la classe mère *(cf. p.7)*

Héritage non complet *Méthode*

Si l'héritage n'est pas complet :

- il faut vérifier la cohérence du type avec les attributs valués
- il faudra gérer la non nullité « à la main »



Héritage presque complet (classe mère abstraite)

`R1(#a,b,c,d,e,f,t:{2,3})`

Contraintes :

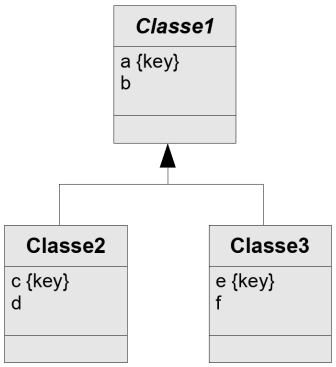
- `t NOT NULL`
- `NOT (t=2 AND f)`
- `NOT (t=3 AND d)`
- `t=2 AND d` (si d est non nul)
- `t=3 AND f` (si f est non nul)

Héritage par la classe mère avec classe mère abstraite

Méthode

Si la classe mère est abstraite :

1. sa valeur est ôtée de l'attribut de discrimination ;
2. une contrainte supplémentaire doit vérifier que soit c soit e est obligatoirement valué.



Héritage (classe mère abstraite)

`R1(#a,b,c,d,e,f,t:{2,3})`

Contraintes :

- `t NOT NULL`
- `c UNIQUE`
- `e UNIQUE`
- `t=2 AND c`
- `t=3 AND e`

- `NOT (t=2 AND (e OR f)`
- `NOT (t=3 AND (c OR d)`

14. Contraintes de l'héritage par les classes filles

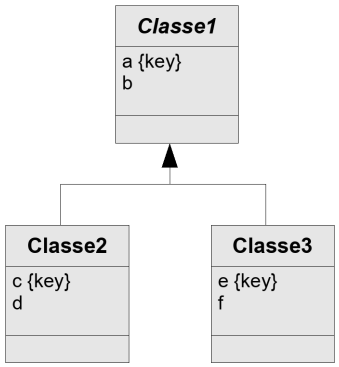
Rappel

Transformation de la relation d'héritage par les classes filles *(cf. p.6)*

Héritage par les classes filles avec clé au niveau de la classe classe mère ***Méthode***

Dès lors que la classe mère possède un attribut unique :

- on ajoute une contrainte pour vérifier cette unicité au niveau des classes filles.



Héritage (classe mère abstraite)

`R2(#a,b,c,d) avec c KEY`

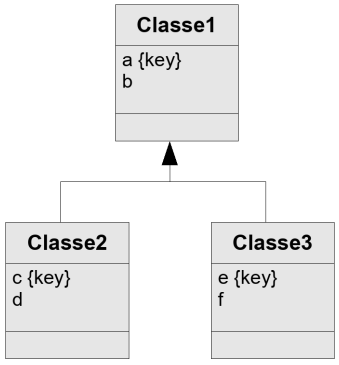
`R3(#a,b,e,f) avec e KEY`

Contraintes : `INTERSECTION (PROJECTION(R2,a), PROJECTION(R3,a)) = {}`

Héritage par les classes filles avec classe mère non abstraite ***Méthode***

Si la classe mère n'est pas abstraite :

- On créé une relation supplémentaire pour gérer les objets de la classe mère
- On ajoute une contrainte qui exprime que les tuples de la classe mère ne peuvent pas exister dans les classes filles



Héritage

`R1(#a,b)`

`R2(#a,b,c,d) avec c KEY`

`R3(#a,b,e,f) avec e KEY`

Contraintes :

- `INTERSECTION (PROJECTION(R2,a), PROJECTION(R3,a)) = {}`
- `INTERSECTION (PROJECTION(R1,a), (PROJECTION(R2,a) UNION PROJECTION(R3,a)) = {}`

Héritage absorbé par les classes filles ? *Exemple*

Soit la classe abstraite A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A avec les attributs B1 et B2. Soit la classe C, classe fille de A avec les attributs C1 et C2.

Le modèle relationnel correspondant selon cette transformation est :

```
B (#K, A1, A2, B1, B2)
C (#K, A1, A2, C1, C2)
Contrainte : INTERSECTION (PROJECTION(B,K), PROJECTION(C,K)) = {}
```

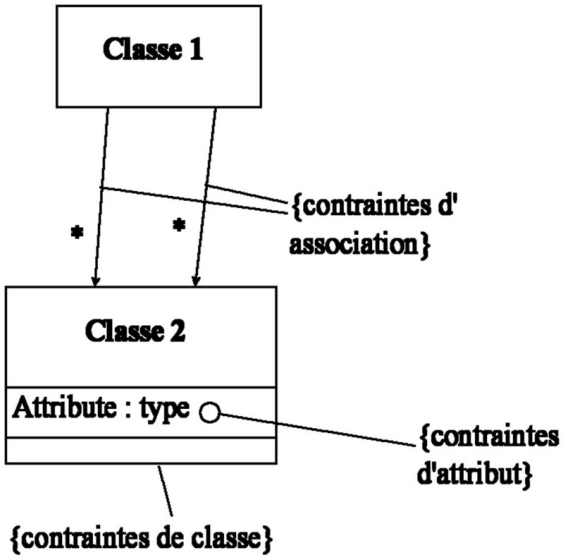
Si A n'avait pas été abstraite elle aurait donné naissance à une relation A possédant les attributs A1 et A2 et qui n'aurait alors contenu que les tuples qui ne sont ni des B ni des C.

15. Contraintes

Ajout de contraintes dynamiques sur le diagramme de classe

Il est possible en UML d'exprimer des contraintes dynamiques sur le diagramme de classe, par annotation de ce dernier.

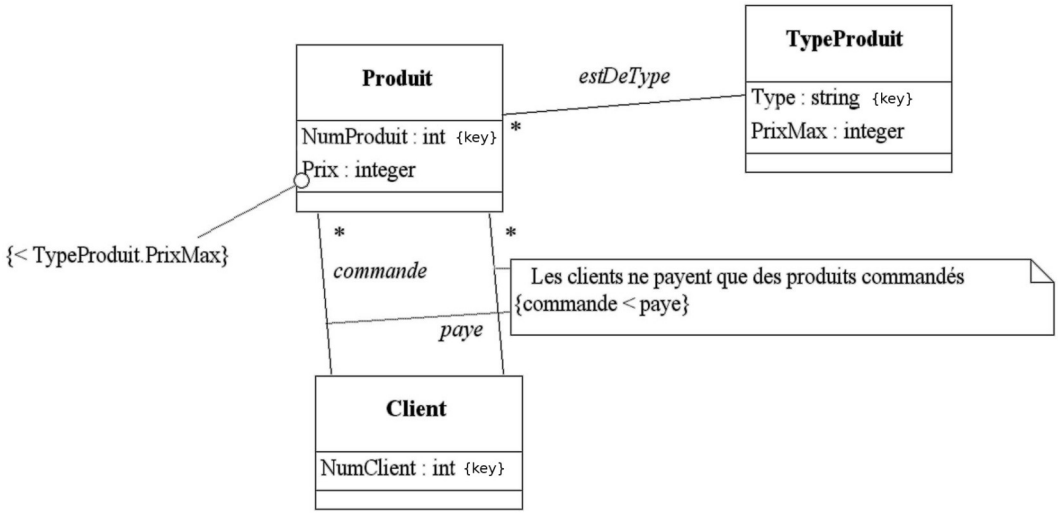
Notation de contraintes § *Syntaxe*



Notation contraintes en UML

Exemple de contraintes

Exemple



Commandes

Expressions formelles ou texte libre ?

Méthode

Il est possible d'exprimer les contraintes en texte libre ou bien en utilisant des expressions formelles.

On privilégiera la solution qui offre le meilleur compromis entre facilité d'interprétation et non ambiguïté. La combinaison des deux est également possible si nécessaire.

Quelles contraintes exprimer ?

Méthode

En pratique il existe souvent de très nombreuses contraintes, dont certaines sont évidentes, ou encore secondaires. Or l'expression de toutes ces contraintes sur un diagramme UML conduirait à diminuer considérablement la lisibilité du schéma sans apporter d'information nécessaire à la compréhension. En conséquence on ne représentera sur le diagramme que les contraintes les plus essentielles.

Notons que lors de l'implémentation toutes les contraintes devront bien entendu être exprimées. Si l'on souhaite préparer ce travail, il est conseillé, lors de la modélisation logique, de recenser de façon exhaustive dans un tableau toutes les contraintes à implémenter.

16. Liste des contraintes

KEY, UNIQUE, NOT NULL

Fondamental

On pensera à exprimer les clés candidates KEY, les attributs UNIQUE et NOT NULL.

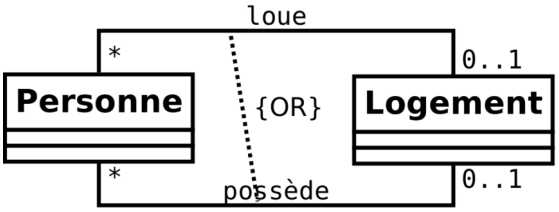
```
1 R1 (#pk, k, u, a) avec k KEY, u UNIQUE, a NOT NULL
```

Contraintes exprimées sur le MCD

Méthode

Les contraintes exprimées au niveau conceptuel doivent être traduites au niveau relationnel.

Si les contraintes ne sont pas trop nombreuses, on peut commenter chaque relation directement lors de la formalisation du MLD.



Exemple de contrainte OR entre deux associations

```
Personne (... loue=>Logement, possède=>Logement) avec (loue OR possède)
Logement (...)
```

Méthode

Si l'expression des contraintes nuit à la lisibilité, on les reportera une liste à la fin du MLD (voire dans un document annexe qui accompagnera le modèle logique).

Extension des contraintes exprimées ***Attention***

On s'attachera lors de la modélisation logique à exprimer l'ensemble des contraintes pesant sur le modèle, même celles qui ont été considérées comme secondaires ou évidentes lors de la modélisation conceptuelle.

Expression des contraintes ***Méthode***

Pour exprimer les contraintes on favorisera un langage formel dans la mesure du possible :

- algèbre relationnelle
- expression logique ou mathématique
- expression SQL ou procédurale
- ...

Immutabilité ***Remarque***

La contrainte {frozen} n'existe pas en base de données relationnelle standard. On ajoutera donc cette contrainte en complément dans le modèle relationnel (et on pourra ajouter un déclencheur au moment de l'implémentation pour empêcher la modification d'un attribut).

Index