Expression de contraintes avancées

Table des matières

Cours

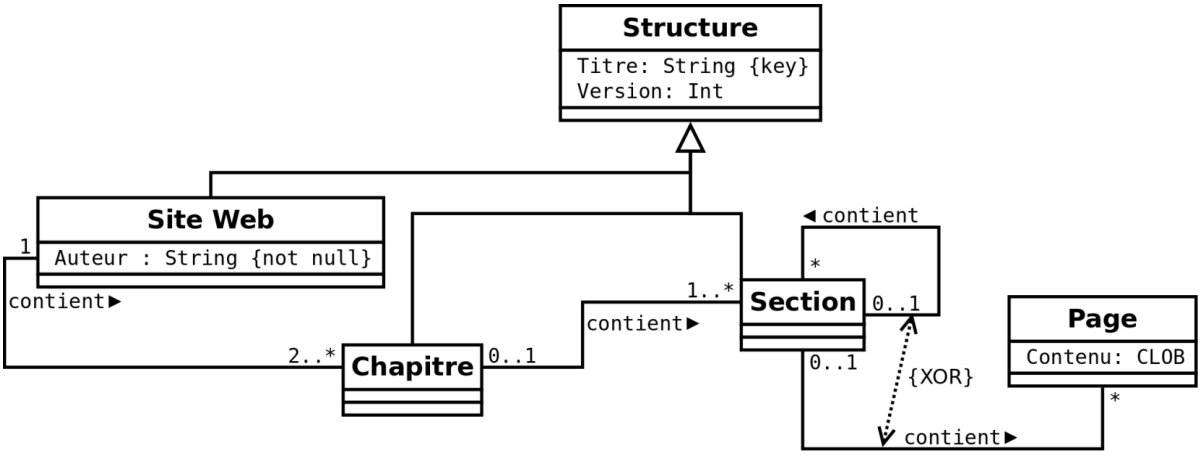
1. Expression de contraintes sur le diagramme de classes

Objectifs

Maîtriser le diagramme de classe UML dans le cas de la conception de BD.

1.1. Exercice : Modéliser un site Web

En analysant le schéma UML ci-après, sélectionner toutes les assertions vraies.



MCD UML

- ☐ Tous les chapitres ont un titre.
- ☐ Il est possible d'avoir un auteur différent pour chaque chapitre.
- ☐ Toutes les sections contiennent au moins une section.
- ☐ Toutes les sections contiennent au moins une page.
- ☐ Toutes les sites Web contiennent au moins une page.

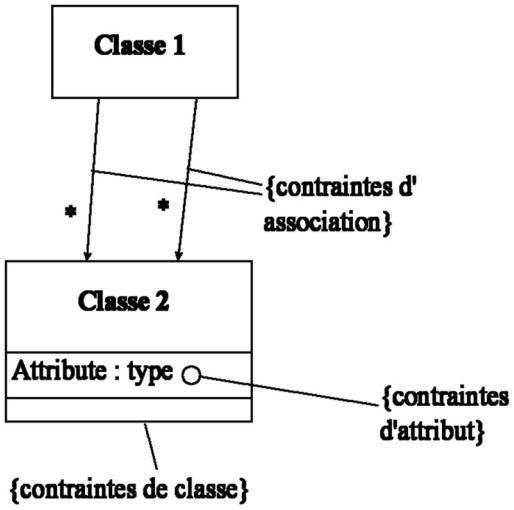
1.2. Contraintes

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

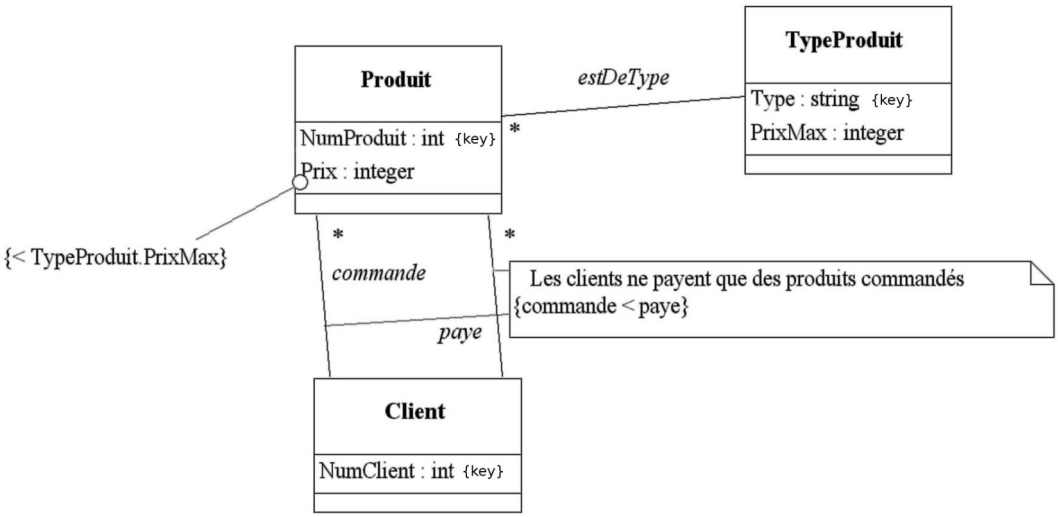
Syntaxe



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.

1.3. Expression de l'optionalité

Optionalité *Syntaxe*

L'attribut a **toujours** une valeur au cours de son cycle de vie, donc dès la création de l'objet auquel il appartient.

Optionalité : [0..1] *Syntaxe*

On considère par défaut en UML que tous les attributs sont non nuls et on indique les attributs optionnels avec la syntaxe `attribut [0..1]`.

Non nullité {not null} *Syntaxe*

Dans un contexte où tous les attributs seraient optionnels sauf exception, on peut adopter l'expression de la non nullité {not null} à la place de l'expression de l'optionalité. On n'utilisera pas les deux syntaxes sur un même schéma UML (sous peine de ne pas savoir comment interpréter les attributs sans [0..1] ni {not null}.

1.4. Expression de l'unicité

Unicité *Définition*

La valeur de l'attribut est unique pour la classe.

{unique} *Syntaxe*

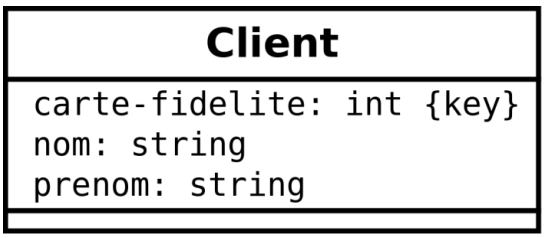
On utilise {unique} pour exprimer l'unicité.

{unique} ou {key} *Syntaxe*

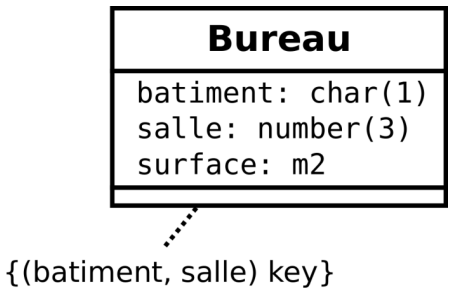
Bien que non standardisé en UML, en base de données, il est courant d'utiliser {key} à la place de {unique}.

- {key} ≡ {unique not null} (à condition que la minimalité soit respectée par ailleurs) ;
- et comme {not null} est une contrainte pas défaut : {unique} ≡ {key} (si l'optionalité n'est pas exprimée et si la contrainte de minimalité est vérifiée)

Contrainte d'unicité *Exemple*



Clé en UML



Clé composée de deux attributs

1.5. Expression de l'immuabilité

Immuabilité — *Définition*

Une fois instancié l'attribut ne peut plus changer.

{frozen} ou {immuable} — *Définition*

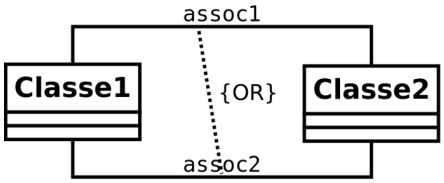
On peut écrire {frozen} ou {immutable}.

Clé primaire — *Rappel*

Pour choisir une clé primaire au niveau relationnel, on privilégiera une clé candidate immuable, et donc une contrainte {unique frozen} (sur un groupe minimal d'attributs non nuls).

1.6. Contraintes standardisées sur les associations

Exemple de notation — *Exemple*

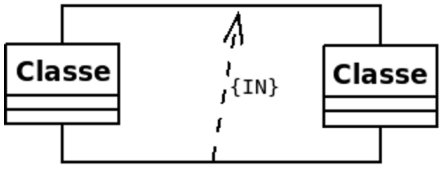


Contrainte OR entre deux associations

Classe1	Classe2	IN(C1→C2)	IN(C2→C1)	AND ou S	X	OR ou T	XOR ou XT
0	0	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	1	0	1	1	1
1	1	1	1	1	0	1	0

Inclusion {I}, également notée {Subset} ou {IN} — *Définition*

Si l'association inclue est instanciée, l'autre doit l'être aussi, la contrainte d'inclusion a un sens, représenté par une flèche.



Contrainte orientée IN

Simultanéité {S}, également notée {=} ou {AND} *Définition*

Si une association est instanciée, l'autre doit l'être aussi.

La simultanéité est équivalente à une double inclusion.

Exclusion {X} *Définition*

Les deux associations ne peuvent être instanciées en même temps.

Totalité {T}, également notée {OR} *Définition*

Au moins une des deux associations doit être instanciée.

Partition {XT}, également notée {+} ou {XOR} ou {P} *Définition*

Exactement une des deux associations doit être instanciée.

2. Passage UML-Relationnel : Expression de contraintes

Objectifs

Savoir exprimer les contraintes en modélisation relationnelle.

2.1. Liste des contraintes

KEY, UNIQUE, NOT NULL *Fondamental*

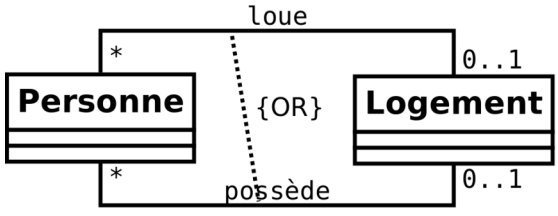
On pensera à exprimer les clés candidates KEY, les attributs UNIQUE et NOT NULL.

```
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*

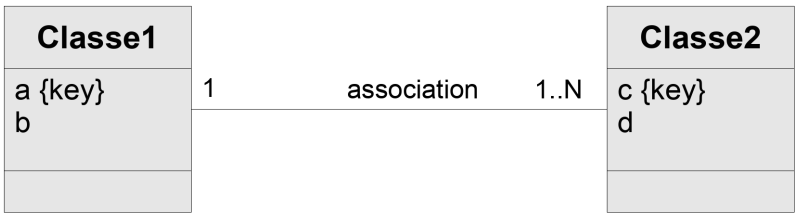
La contrainte {frozen} n'existe pas en base de données relationnelle standard. On ajoutera donc cette contrainte en complément dans le modèle relationnel (et on pourra ajouter un déclencheur au moment de l'implémentation pour empêcher la modification d'un attribut).

2.2. Contrainte de cardinalité minimale 1 dans les associations 1:N

Rappel

Transformation des associations 1:N *(cf. p.17)*

Méthode



Association 1:N

- Si la cardinalité est exactement 1 (1..1) côté 1, alors on ajoutera une contrainte de non nullité sur la clé étrangère,
- si la cardinalité est au moins 1 (1..N) côté N, on ajoutera une contrainte d'existence de tuples référençant pour chaque tuple de la relation référencée.

```
R1(#a,b)
```

```
R2(#c,d,a=>R1)
```

Contraintes : `R2.a NOT NULL et PROJECTION(R1,a) = PROJECTION(R2,a)`

On veut que tous les éléments de R1 soient référencés au moins une fois dans R2, donc il n'existe pas de tuple de R1 qui ne soit pas référencé par R2.a, donc `PROJECTION(R1,a) = PROJECTION(R2,a)`.

R1

#a	b
1	Lorem
2	Ipsum

R2

#c	b	a=>R1
a	Sed	1
b	Ut	2
c	Perspiciatis	2

Cas général : PROJECTION(R1,a) ⊆ PROJECTION(R2,a) ***Complément***

Si la cardinalité côté 1 est 0..1 alors il peut exister la valeur NULL dans R2.a et donc la contrainte devient : `PROJECTION(Classe1,a) ⊆ PROJECTION(Classe2,a).`

R1

#a	b
1	Lorem
2	Ipsum

R2

#c	b	a=>R1
a	Sed	1
b	Ut	2
c	Perspiciatis	2
d	Unde	NULL

La projection élimine les doublons ***Rappel***

Projection *(cf. p.17)*

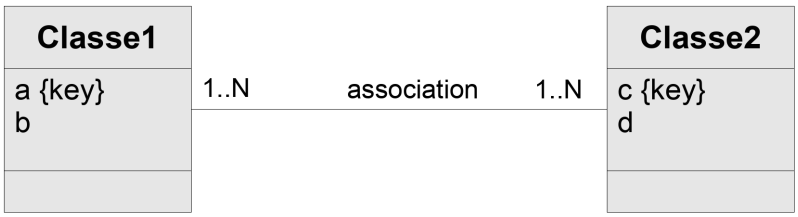
2.3. Contrainte de cardinalité minimale 1 dans les associations N:M

Rappel

Transformation des associations N:M *(cf. p.18)*

Méthode

Si la cardinalité est au moins 1 (1..N) d'un côté et/ou de l'autre, alors des contraintes d'existence simultanée de tuples devront être ajoutées.



Association N:M

`R1(#a,b)`

`R2(#c,d)`

`A(#a=>R1,#c=>R2)`

Contraintes : `PROJECTION(R1,a) = PROJECTION(A,a) AND PROJECTION(R2,c) = PROJECTION(A,c)`

Remarque

Si la cardinalité est `0..N:1..N` seule une des deux contraintes est exprimée.

La projection élimine les doublons

Rappel

Projection *(cf. p.17)*

2.4. Contraintes de l'héritage par référence

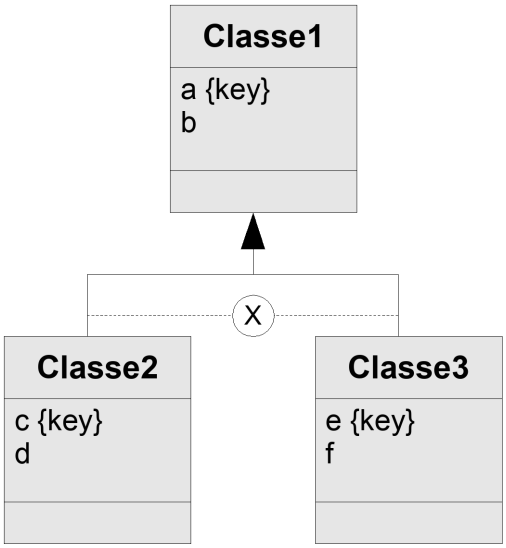
Rappel

Transformation de la relation d'héritage par référence *(cf. p.19)*

Héritage exclusif (cas général)

Méthode

Dans le cas général d'un héritage exclusif, il faut ajouter une contrainte pour que les même tuples ne se trouvent pas à la fois dans R2 et R3



Héritage exclusif (non complet)

`R1(#a,b)`

`R2(#a=>R1,c,d) avec c KEY`

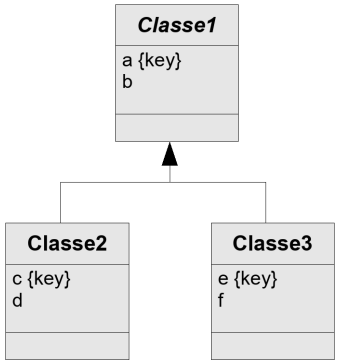
`R3(#a=>R1,e,f) avec e KEY`

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

Héritage par référence avec classe mère abstraite

Méthode

Si la classe mère est abstraite, il faut ajouter la contrainte que tous les tuples de R1 sont référencés par un tuple de R2 ou de R3.



Héritage (classe mère abstraite)

`R1(#a,b)`

`R2(#a=>R1,c,d) avec c KEY`

`R3(#a=>R1,e,f) avec e KEY`

Contraintes : `PROJECTION(R1,a) = PROJECTION(R2,a) UNION PROJECTION(R3,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)
```

Cette solution est particulièrement adaptée lorsque la classe mère n'est pas abstraite, car cela en autorise l'instanciation sans aucun bruit dans la relation lié aux classes filles. Par contre si la classe mère est abstraite il faut introduire une contrainte dynamique complexe pour imposer la présence d'un tuple référençant dans une des classes filles.

Ainsi dans l'exemple précédent, un A peut être instancié en toute indépendance par rapport à la relation B.

2.5. Contraintes de l'héritage par les classes filles

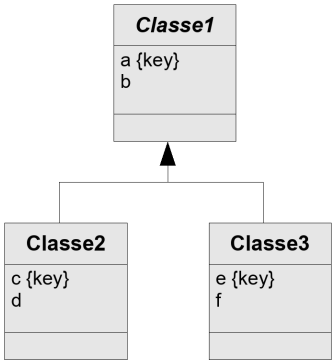
Rappel

Transformation de la relation d'héritage par les classes filles *(cf. p.22)*

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)

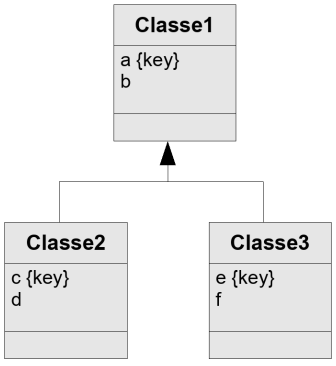
```
R2(#a,b,c,d) avec c KEY
R3(#a,b,e,f) avec e KEY
```

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 :

```
1 B (#K, A1, A2, B1, B2)
2 C (#K, A1, A2, C1, C2)
3 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.

2.6. Contraintes de l'héritage par la classe mère

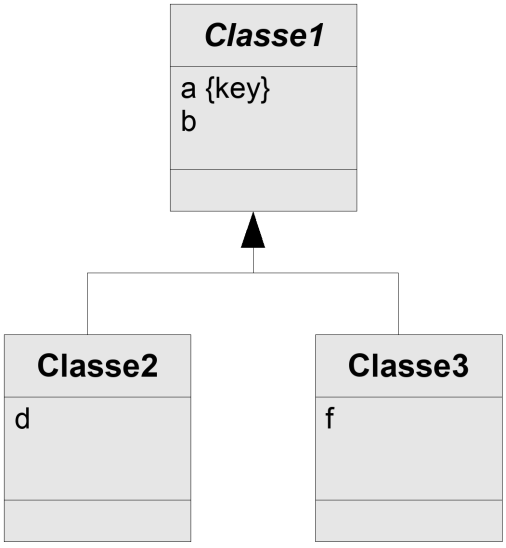
Rappel

Transformation de la relation d'héritage par la classe mère *(cf. p.26)*

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})
```

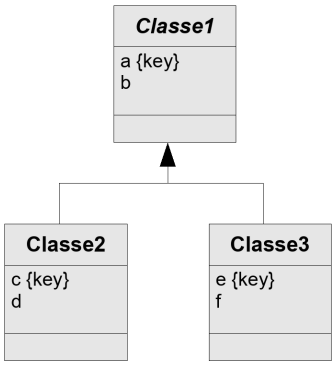
Contraintes :

- `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)`

2.7. 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.

Exercices

1. Exercice : Médiathèque

[60 minutes]

Une médiathèque veut s'informatiser pour gérer plus efficacement son catalogue, contenant des livres, des films, et des albums de musique.

Les informations à enregistrer pour les livres sont : les titres, le (ou les) auteur(s), la date de parution, l'éditeur, la date d'impression et le nombre de pages. Les informations à enregistrer pour les films sont : le nom, le (ou les) réalisateur(s), au maximum six acteurs ayant joué dans le film avec leur rôle (il peut n'y avoir aucun acteur), le studio de production et la date de sortie. Les informations à gérer pour les albums sont : le nom de l'album, l'auteur, la maison de production, l'année de sortie, et les supports disponibles à la médiathèque (CD et/ou vinyle).

Pour chaque auteur, réalisateur, acteur ou musicien d'une œuvre, on veut gérer son nom, son prénom, son ou ses noms d'artiste (s'il en a), et sa date de naissance. Un album de musique peut avoir pour auteur :

- un artiste (ex : Jimmy Hendrix, Mozart),
- ou un groupe (ex : les Rolling Stones ou Franz Ferdinand) dont on gérera le nom, l'année de formation, et les informations sur les différents membres.

Livres, films et albums sont identifiés de manière unique (un film ne peut pas avoir le même identifiant qu'un album) par un code interne à la médiathèque. On veut aussi pouvoir gérer les adaptations cinématographiques (quels films sont les adaptations de quels livres). On veut enfin pouvoir retrouver facilement la BO correspondant à un film donné (la bande originale est l'album de musique du film).

Question 1

Réalisez un *package* permettant de modéliser en UML le catalogue de la médiathèque.

La médiathèque s'étend sur trois étages, chacun comprenant plusieurs rayons à thème (Romans Contemporains, Musique Classique, etc.). Certains de ces rayons peuvent comprendre à la fois des albums, des livres et des films (comme le rayon Science Fiction, ou le rayon Thriller). Dans un rayon, les œuvres centrales ou récentes sont disposées sur des présentoirs (chacun pouvant recueillir un certain nombre d'œuvres : films, livres et/ou albums), tandis que les autres sont rangées sur des étagères identifiées par un code particulier. Chaque étagère peut comprendre un certain nombre d'œuvres, mais d'une seule catégorie (on a des étagères de livres et des étagères d'albums, mais pas d'étagères comprenant à la fois des livres et des albums).

Question 2

Réalisez un second *package* permettant de modéliser l'organisation de la bibliothèque et de ses moyens de rangement.

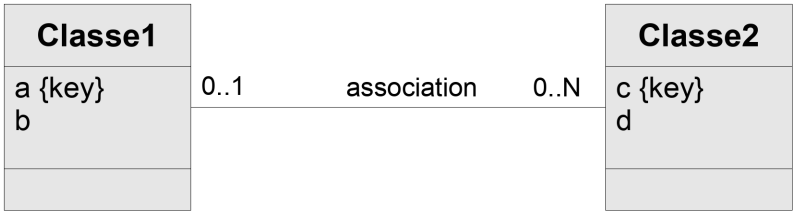
Contenus annexes

1. Transformation des associations 1:N

Méthode

Pour chaque association binaire de type 1:N :

- on ajoute à la relation côté N une clé étrangère vers la relation côté 1.



Association 1:N

```
Classe1(#a,b)
Classe2(#c,d,a=>Classe1)
```

Complément

Contrainte de cardinalité minimale 1 dans les associations 1:N *(cf. p.8)*

2. 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, ...)`

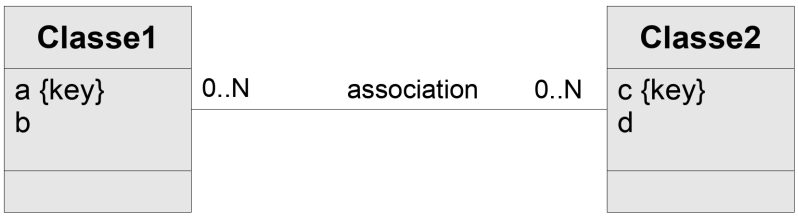
$R = \pi_{a1,a2,...}(R1)$

3. Transformation des associations N:M

Méthode

Pour chaque association binaire de type M:N :

- on crée une nouvelle relation,
- composée de clés étrangères vers chaque relation associée,
- et dont la clé primaire est la concaténation de ces clés étrangères.



Association N:M

```
Classe1(#a,b)
Classe2(#c,d)
Assoc(#a=>Classe1,#c=>Classe2)
```

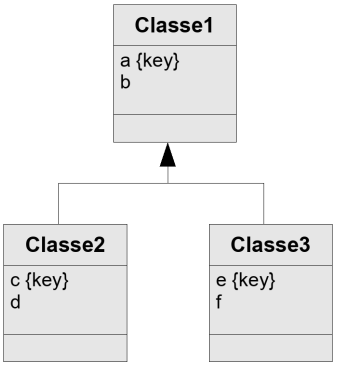
Complément

Contrainte de cardinalité minimale 1 dans les associations N:M *(cf. p.9)*

4. 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.20)*

Méthode alternative

Complément

Transformation de la relation d'héritage par référence (de la mère vers les filles) *(cf. p.22)*

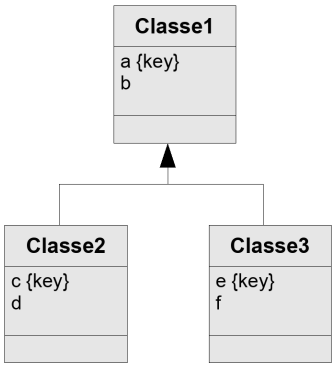
5. Héritage par une référence et vues

Rappel

Transformation de la relation d'héritage par référence *(cf. p.19)*

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.20)*

6. 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.

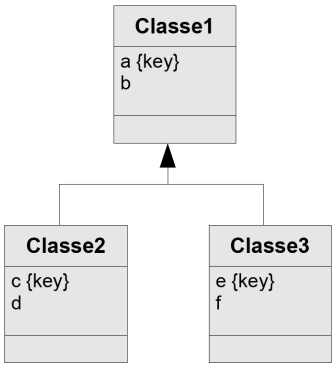
Syntaxes alternatives *Complément*

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

7. 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



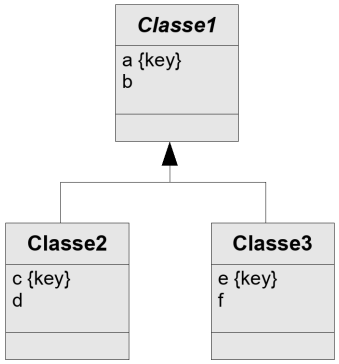
Héritage

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

8. 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

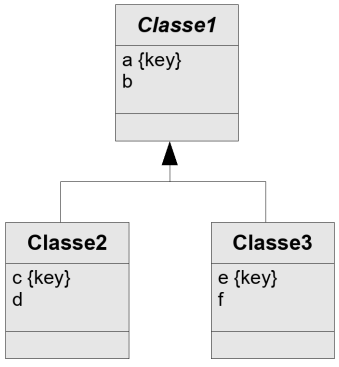
Héritage par les classes filles et vues (cf. p.23)

9. Héritage par les classes filles et vues

Rappel

Transformation de la relation d'héritage par les classes filles (cf. p.22)

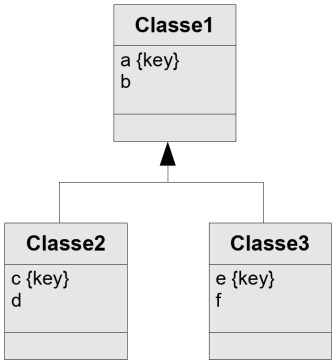
Héritage absorbé par les classes filles (classe mère abstraite) — **Méthode**



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.24)*

Projection *(cf. p.17)*

10. 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`

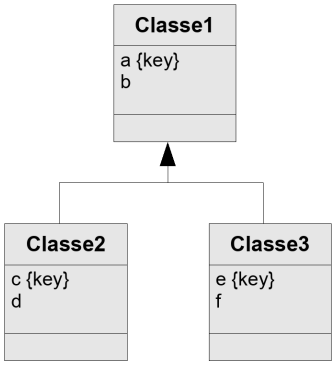
`R = R1 ∩ R1`

`R = R1 − R2`

11. 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.27)

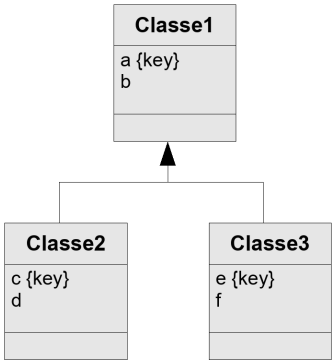
12. Héritage par la classe mère et vues

Rappel

Transformation de la relation d'héritage par la classe mère (cf. p.26)

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.17)*

Restriction *(cf. p.28)*

13. 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.

Index