Vues et gestion des droits

Table des matières

Cours

1. Notion de schéma externe et de vue

1.1. Exercice : Problème de vue

Soit la vue V suivante, sélectionner **toutes** les assertions correctes.

```
1 CREATE VIEW V (n, p) AS SELECT nom, prenom FROM T ;
```

- ☐ Le contenu de la table T est calculé dynamiquement à partir de la vue V.
- ☐ Le contenu de la table T est stocké directement dans la base de données.
- ☐ Le contenu de la vue V est calculé dynamiquement à partir de la table T.
- ☐ Le contenu de la vue V est stocké directement dans la base de données.
- ☐ La requête `SELECT n FROM V` est valide.
- ☐ La requête `SELECT nom FROM V` est valide.
- ☐ L'instruction `CREATE VIEW V2 (n) AS SELECT n FROM V` est valide.

1.2. Schéma externe

L'architecture ANSI/SPARC est à l'origine de la conception des SGBDR, elle postule deux principes fondamentaux : la séparation logique/physique et la notion de schéma externe.

La séparation du niveau logique et du niveau physique *Rappel*

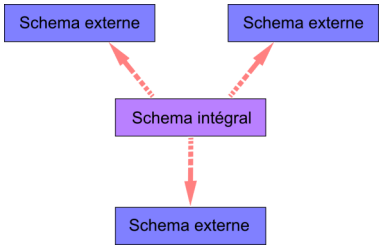
L'utilisateur du SGBDR ne se préoccupe pas de la façon dont celui-ci est implémenté, il raisonne directement en relationnel grâce au langage SQL.

Notion de schéma externe

Le schéma relationnel de la base de données intègre **toutes** les données que la base gère pour **toutes** les applications d'une organisation. Or chaque application n'utilise en général qu'une partie de ces données.

On souhaite que chaque application ne **voit** que la partie des données qui la concerne :

- pour des raisons de simplicité (la complexité globale lui est masquée)
- pour des raisons de sécurité (on ne souhaite pas qu'une application accède à des données qui ne la concerne pas).



Schémas interne et externes

Définition

On appelle schéma externe un sous-ensemble du schéma intégral, destiné à une utilisation spécifique de la base de données.

Dans les SGBDR les schémas externes sont implémentés par des vues.

1.3. Création de vues en SQL (CREATE VIEW)

Vue

Définition

Une vue est une définition logique d'une relation, sans stockage de données, obtenue par interrogation d'une ou plusieurs tables de la *BD*∗. Une vue peut donc être perçue comme une fenêtre dynamique sur les données, ou encore une requête stockée (mais dont seule la définition est stockée, pas le résultat, qui reste calculé dynamiquement).

Une vue permet d'implémenter le concept de schéma externe d'un modèle conceptuel.

Synonymes : Relation dérivée, Table virtuelle calculée

Syntaxe

```
CREATE VIEW <nom de vue> <nom des colonnes>
AS <spécification de question>
```

La spécification d'une question se fait en utilisant le *LMD*∗.

Le nombre de colonnes nommées doit être égal au nombre de colonnes renvoyées par la question spécifiée. Le nom des colonnes est optionnel, s'il n'est pas spécifié, c'est le nom des colonnes telle qu'elles sont renvoyées par la question, qui sera utilisé.

Exemple

```
CREATE VIEW Employe (Id, Nom)
AS
  SELECT N°SS, Nom
  FROM Personne
```

La vue Employe est ici une projection de la relation Personne sur les attributs N°SS et Nom, renommés respectivement Id et Nom.

Vue en lecture et vue en écriture

Remarque

Une vue est toujours disponible en lecture, à condition que l'utilisateur ait les droits spécifiés grâce au *LCD*∗. Une vue peut également être disponible en écriture dans certains cas, que l'on peut restreindre aux cas où la question ne porte que sur une seule table (même si dans certains cas, il est possible de modifier une vue issue de plusieurs tables).

Dans le cas où une vue est destinée à être utilisée pour modifier des données, il est possible d'ajouter la clause "WITH CHECK OPTION" après la spécification de question, pour préciser que les données modifiées ou ajoutées doivent effectivement appartenir à la vue.

Vue sur une vue — *Remarque*

Une vue peut avoir comme source une autre vue.

Vues et héritage — *Rappel*

Les vues sont particulièrement utiles pour restituer les relations d'héritage perdues lors de la transformation *MCD∗* vers *MLD∗*.

2. Passage UML-Relationnel : Expression des vues pour l'héritage et les méthodes

Objectifs

Savoir ajouter des vues pour améliorer l'expression de l'héritage en relationnel.

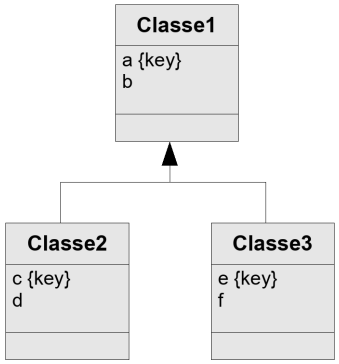
2.1. Héritage par une référence et vues

Rappel

Transformation de la relation d'héritage par référence *(cf. p.15)*

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 :

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

Algèbre relationnel

Complément

Jointure *(cf. p.16)*

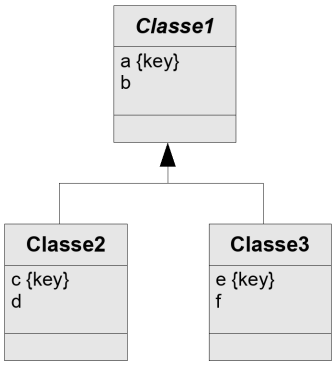
2.2. Héritage par les classes filles et vues

Rappel

Transformation de la relation d'héritage par les classes filles *(cf. p.18)*

Héritage absorbé par les classes filles (classe mère abstraite)

Méthode

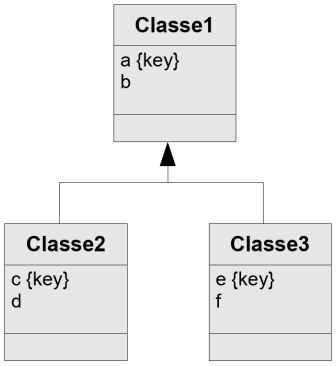


Héritage (classe mère abstraite)

```
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.18)*

Projection *(cf. p.20)*

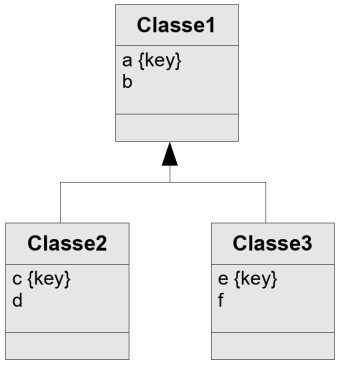
2.3. Héritage par la classe mère et vues

Rappel

Transformation de la relation d'héritage par la classe mère *(cf. p.21)*

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.20)*

Restriction *(cf. p.22)*

2.4. Transformation des méthodes par des vues

Méthode

Lorsqu'une méthode est spécifiée sur un diagramme UML pour une classe C, si cette méthode est une fonction relationnelle (elle renvoie une unique valeur et elle peut être résolue par une requête SQL), alors on crée une vue qui reprend les attributs de la classe C et ajoute des colonnes calculées pour les méthodes.

Attributs dérivés — *Remarque*

Les attributs dérivés étant apparentés à des méthodes, ils peuvent également être gérés par des vues.

2.5. Exercice : Évaluation des enseignants

A la fin du semestre, chaque enseignant est évalué par les étudiants pour chacune de ses UV. Chaque *note* correspond à une UV assurée par cet enseignant, et est égale à la moyenne des *évaluations* attribuées par les étudiants de l'UV. Dans le relevé de note apparaît une *appréciation* générale sur l'enseignant. Cette appréciation est la moyenne de toutes les notes de l'enseignant pour toutes ses UV. La figure suivante illustre le diagramme de classe et le modèle relationnel associé, qui modélisent l' « évaluation des enseignants ».

Les notes sont sur 20.

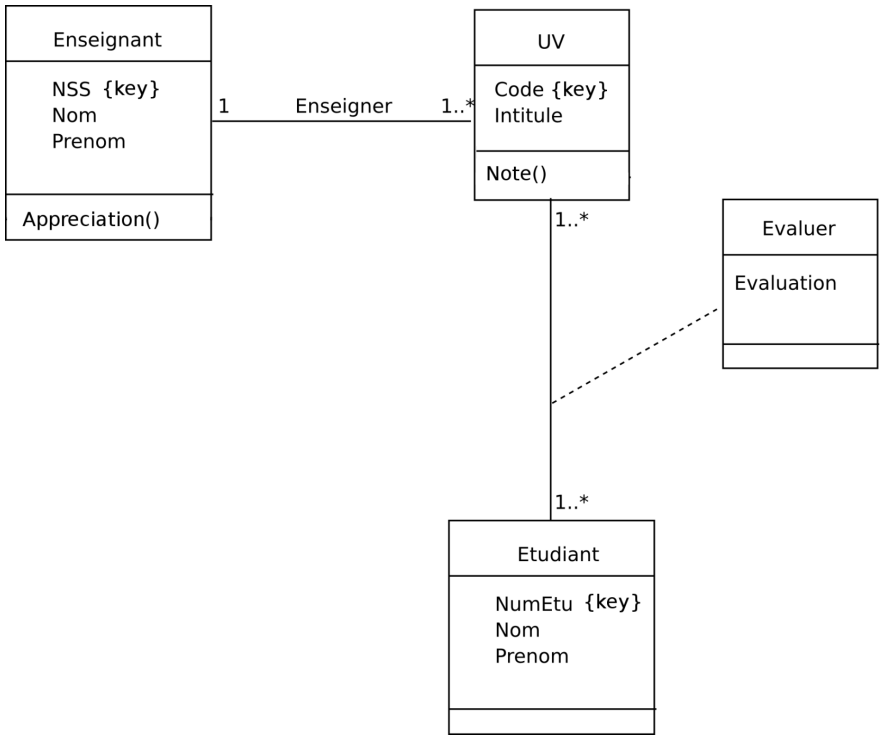


Diagramme de classe

```
Enseignant(#NSS:char(13), Nom:varchar, Prenom:varchar)
UV(#Code:char(5), Intitulé:varchar, Prof=>Enseignant) avec Initulé KEY
Etudiant(#NumEtu:char(20), Nom:varchar, Prenom:varchar)
Evaluer(#NumEtu=>Etudiant, #uv=>UV, Evaluation:[0..20])
```

Question 1

Écrire la vue SQL qui permet de calculer les notes de chaque UV et donnant le résultat ci-dessous : méthode UV.Note()

NOM	PRENOM	INTITULE	NOTE
Dupont	Jean	Science de la Terre	15
Dupont	Jean	Informatique	7.5
Durand	Paul	Français	5

Notes pour chaque UV

Question 2

Écrire la vue SQL qui permet d'afficher les enseignants avec leur appréciation générale : méthode Enseignant.Appreciation()

NOM	PRENOM	APPRECIATION
Dupont	Jean	11,25
Durand	Paul	5

Appréciation pour chaque enseignant

3. Le Langage de Contrôle de Données de SQL

Objectifs

Maîtriser les bases du SQL pour attribuer et révoquer des droits sur des objets d'une base de données.

Le *LCD∗* permet de créer les utilisateurs et de définir leurs droits sur les objets de la *BD∗* de façon déclarative. Il permet notamment l'attribution et la révocation de droits à des utilisateurs, sur l'ensemble des bases du SGBD, sur une BD en particulier, sur des relations d'une BD, voire sur certains attributs seulement d'une relation.

3.1. Exercice : Les droits Lambda

Quelles sont les instructions SQL (sous-ensemble LMD) autorisées pour l'utilisateur "Lambda", étant données les instructions SQL (sous-ensemble LCD) suivantes, exécutées antérieurement ?

```
REVOKE ALL PRIVILEGES ON * FROM Lambda;
GRANT UPDATE, SELECT ON a TO Lambda;
GRANT SELECT ON b TO Lambda;
```

- ☐ SELECT a.x, b.y
  FROM a,b
  WHERE a.x=b.x
- ☐ UPDATE b
  SET y='y'
  WHERE y='x'
- ☐ UPDATE a
  SET x='x'
  WHERE x='y'

- ☐ CREATE TABLE c (
  x CHAR(50),
  y CHAR(50))
- ☐ SELECT a.x, c.y
  FROM a,c
  WHERE a.x=c.x
- ☐ INSERT INTO a (x)
  VALUES ('y')

3.2. Attribution de droits

SQL propose une commande pour attribuer des droits à des utilisateurs sur des tables.

§ Syntaxe

```
GRANT <liste de droits> ON <nom table> TO <utilisateur> [WITH GRANT OPTION]
```

Les droits disponibles renvoient directement aux instructions SQL que l'utilisateur peut exécuter :

- SELECT
- INSERT
- DELETE
- UPDATE
- ALTER

De plus il est possible de spécifier le droit ALL PRIVILEGES qui donne tous les droits à l'utilisateur (sauf celui de transmettre ses droits).

La clause WITH GRANT OPTION est optionnelle, elle permet de préciser que l'utilisateur a le droit de transférer ses propres droits sur la table à d'autres utilisateur. Une telle clause permet une gestion décentralisée de l'attribution des droits et non reposant uniquement dans les mains d'un administrateur unique.

La spécification PUBLIC à la place d'un nom d'utilisateur permet de donner les droits spécifiés à tous les utilisateurs de la *BD*∗.

? Exemple

```
GRANT SELECT, UPDATE ON Personne TO Pierre;
GRANT ALL PRIVILEGES ON Adresse TO PUBLIC;
```

Droits sur une vue ***Remarque***

Il est possible de spécifier des droits sur des vues plutôt que sur des tables, avec une syntaxe identique (et un nom de vue à la place d'un nom de table).

Catalogue de données ***Remarque***

Les droits sont stockés dans le catalogue des données, il est généralement possible de modifier directement ce catalogue à la place d'utiliser la commande GRANT. Cela reste néanmoins déconseillé.

Création des utilisateurs — *Remarque*

Les modalités de création d'utilisateurs, voire de groupes d'utilisateurs dans le SGBD, reste dépendantes de celui-ci. Des commande SQL peuvent être disponibles, telles que CREATE USER, ou bien la commande GRANT lorsque qu'elle porte sur un utilisateur non existant peut être chargée de créer cet utilisateur. Des modules spécifiques d'administration sont généralement disponibles pour prendre en charge la gestion des utilisateurs.

3.3. Révocation de droits

SQL propose une commande pour révoquer les droits attribués à des utilisateurs.

Syntaxe

```
REVOKE <liste de droits> ON <nom table> FROM <utilisateur>
```

Exemple

```
REVOKE SELECT, UPDATE ON Personne FROM Pierre;
REVOKE ALL PRIVILEGES ON Adresse FROM PUBLIC;
```

Révocation du droit de donner les droits — *Remarque*

Pour retirer les droits de donner les droits à un utilisateur (qui l'a donc obtenu par la clause WITH GRANT OPTION), il faut utiliser la valeur GRANT OPTION dans la liste des droits révoqués.

Révocation en cascade — *Remarque*

Lorsque qu'un droit est supprimé pour un utilisateur, il l'est également pour tous les utilisateurs qui avait obtenu ce même droit par l'utilisateur en question.

3.4. Création d'utilisateurs

Le standard SQL laisse la gestion des utilisateurs à la discrétion du SGBD.

Néanmoins le commande `CREATE USER` est couramment proposée (Oracle, Postgres, ...).

3.5. Exercice : The show must go on

[10 minutes]

Soit la base de données suivantes :

```
SPECTACLE (#nospectacle:int, nom:str, durée:minutes, type:{théâtre|danse|concert})
SALLE (#nosalle:int, nbplaces:int)
REPRESENTATION (#date:timestamp, #nospectacle=>SPECTACLE, #nosalle=>SALLE,
prix:decimal)
```

On suppose des classes d'utilisateurs qui ont accès à tout ou partie de ce schéma relationnel :

- Le programmateur qui entre les spectacles dans la base de données,
- Le régisseur qui gère les salles et les représentations,
- Les clients qui peuvent accéder au programme.

Question

Donner les droits associés à chaque classe d'utilisateurs.

Exercice

1. Exercice : Du producteur au consommateur++

[30 min]

Soit la base de données suivante :

```
CREATE TABLE  Producteur (
raison_sociale VARCHAR (25),
ville VARCHAR(255),
PRIMARY KEY (raison_sociale)
);

CREATE TABLE Consommateur (
login VARCHAR(10),
email VARCHAR(50),
nom VARCHAR(50) NOT NULL,
prenom VARCHAR(50) NOT NULL,
ville VARCHAR(255) NOT NULL,
PRIMARY KEY (login,email),
UNIQUE (nom,prenom,ville)
);

CREATE TABLE Produit (
id INTEGER,
description VARCHAR(100),
produit_par VARCHAR(25) NOT NULL,
consomme_par_login VARCHAR(10),
consomme_par_email VARCHAR(50),
PRIMARY KEY (id),
FOREIGN KEY (produit_par) REFERENCES Producteur(raison_sociale),
FOREIGN KEY (consomme_par_login,consomme_par_email) REFERENCES
Consommateur(login,email)
);
```

Question 1

Établissez les instructions LCD permettant d'attribuer :

- les droits en lecture seule pour tous les utilisateurs pour la table `Produit`
- les droits en lecture et en écriture pour l'utilisateur `Admin` sur toutes les tables.

Question 2

Afin d'alimenter une application de suivi nommée *Big Brother* écrivez les trois vues SQL LMD permettant de connaître :

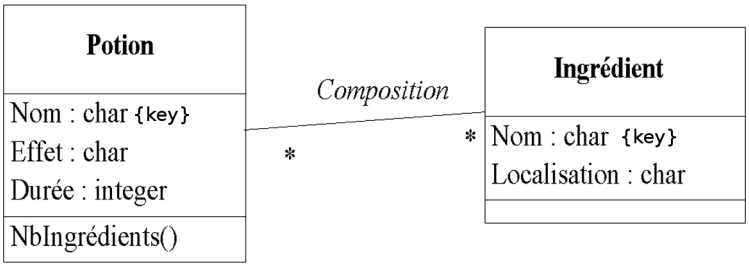
- Les produits produits et consommés dans la même ville
- Les produits qui ne sont pas consommés
- Le nombre de produits produits par chaque producteur

Établissez un schéma externe limité à ces trois vues pour l'utilisateur `BB` (sous lequel se connecte l'application *Big Brother*).

2. Exercice : Gauloiseries

[30 minutes]

Un vieux druide perdant la mémoire souhaite réaliser une base de données pour se souvenir de la composition de ses potions. Un de ses apprentis ayant suivi un cours sur les bases de données lors de son initiation druidique, il réalise le schéma conceptuel suivant :



Potion : modèle UML

Vous pouvez tester vos requêtes directement sur votre base de données en l'initialisant avec le fichier de données.

```sql
/**
DROP TABLE Composition ;
DROP TABLE Ingredient ;
DROP TABLE Potion ;
**/

CREATE TABLE Potion (
  Nom VARCHAR PRIMARY KEY,
  Effet VARCHAR,
  Duree INTEGER
);

CREATE TABLE Ingredient (
  Nom VARCHAR PRIMARY KEY,
  Localisation VARCHAR
);

CREATE TABLE Composition (
  NomP VARCHAR REFERENCES Potion(Nom),
  NomI VARCHAR REFERENCES Ingredient(Nom),
  PRIMARY KEY (NomP, NomI)
);

INSERT INTO Potion (Nom, Effet, Duree) VALUES ('Potion Magique','Force', 60);
INSERT INTO Ingredient (Nom, Localisation) VALUES ('Eau', 'Partout');
INSERT INTO Ingredient (Nom, Localisation) VALUES ('Gui', 'Forêt');
INSERT INTO Ingredient (Nom, Localisation) VALUES ('Pomme', 'Pommier');
INSERT INTO Composition (NomP, NomI) VALUES ('Potion Magique', 'Eau');
INSERT INTO Composition (NomP, NomI) VALUES ('Potion Magique', 'Gui');
INSERT INTO Composition (NomP, NomI) VALUES ('Potion Magique', 'Pomme');
INSERT INTO Ingredient (Nom, Localisation) VALUES ('Bière', 'Pic');
INSERT INTO Potion (Nom, Effet, Duree) VALUES ('Potion Inutile','Rien', 0);
INSERT INTO Composition (NomP, NomI) VALUES ('Potion Inutile', 'Eau');
INSERT INTO Potion (Nom, Effet, Duree) VALUES ('Eau Aromatisée','Goût', 10);
INSERT INTO Composition (NomP, NomI) VALUES ('Eau Aromatisée', 'Eau');
INSERT INTO Composition (NomP, NomI) VALUES ('Eau Aromatisée', 'Gui');
```

Question 1

Réaliser le modèle logique correspondant en relationnel, en faisant apparaître les clés primaires et étrangères (sans clé artificielle).

Question 2

Écrivez une requête SQL permettant de trouver la recette de la potion magique.

Question 3

Qu'a-t-on gagné à ne pas avoir utilisé de clé artificielle dans notre modèle relationnel ?

Question 4

Écrivez une requête SQL permettant de trouver les ingrédients utilisés par aucune potion.

Question 5

Écrivez une vue SQL permettant de préparer l'implémentation de la méthode *NbIngrédients*. Cette vue devra renvoyer le nombre d'ingrédients pour chaque potion.

Question 6

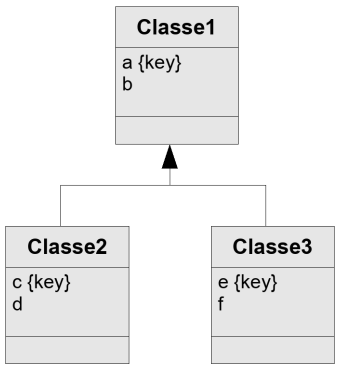
Critiquez le schéma conceptuel de l'apprenti : Est-ce qu'un même ingrédient peut apparaître deux fois dans la même potion ? Peut-on gérer la quantité d'ingrédients pour chaque potion ? Proposer une solution et redessiner le schéma conceptuel en Entité-Association, en y intégrant votre amélioration.

Contenus annexes

1. 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.5)*

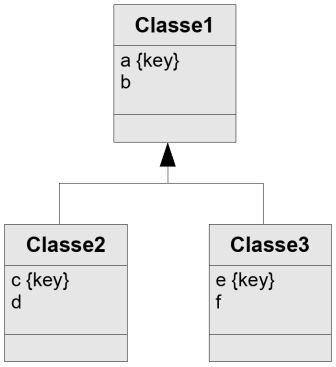
Méthode alternative *Complément*

Transformation de la relation d'héritage par référence (de la mère vers les filles) (cf. p.16)

2. 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)
```

3. 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

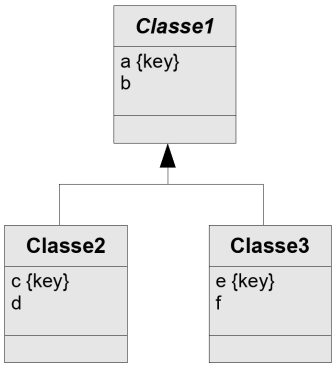
`R = ⋈ (R1, R2, condition)`

`R = R1 ⋈(condition) R2`

4. 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.6)

5. 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
```

6. 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***

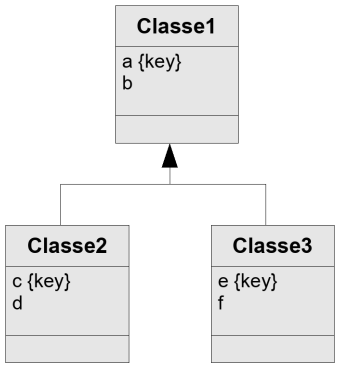
`R = π (R1, a1, a2, ...)`

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

7. 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.7)

8. 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.

Abréviations

BD : Base de Données
LCD : Langage de Contrôle de Données
LMD : Langage de Manipulation de Données
MCD : Modèle Conceptuel de Données
MLD : Modèle Logique de Données

Index