stph.scenari-community.org/bdd

Conception des bases de données II

Conception des bases de données relationnelles

bdd2.pdf

utc Formation

Stéphane Crozat

Publiée le 14 février 2017

Table des matières

L'héritage dans la modélisation conceptuelle de données

I

A. Cours

1. Introduction à l'héritage

Objectifs

Savoir utiliser l'héritage lors une modélisation

a) Exercice : Mariages

Étant donné le schéma UML suivant, quelles sont les assertions vraies ?

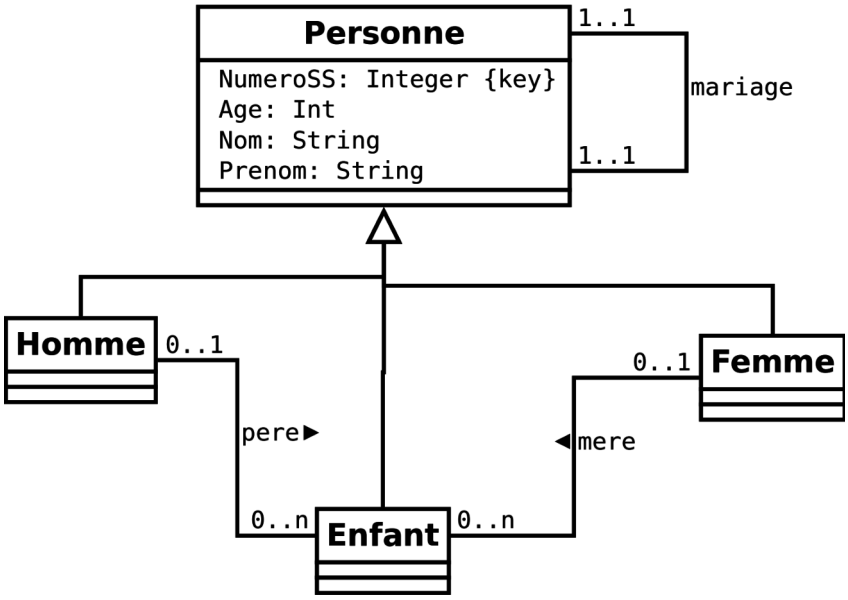


Image 1

☐	Les mariages homosexuels sont possibles.
☐	La polygamie est possible.
☐	Une femme peut ne pas être mariée.
☐	Les enfants peuvent être plus âges que leurs parents.
☐	Deux hommes peuvent avoir un enfant ensemble.
☐	Une femme peut avoir plusieurs enfants.
☐	Un enfant a obligatoirement deux parents.
☐	Les enfants peuvent se marier.
☐	Tous les enfants sont mariés.
☐	Les personnes mariées ont toujours le même nom.

b) Héritage

Définition : Héritage

L'héritage est l'association entre deux classes permettant d'exprimer que l'une est plus générale que l'autre. L'héritage implique une transmission automatique des propriétés (attributs et méthodes) d'une classe A à une classe A'.

Dire que A' hérite de A équivaut à dire que A' est une sous-classe de A. On peut également dire que A est une généralisation de A' et que A' est une spécialisation de A.

Syntaxe

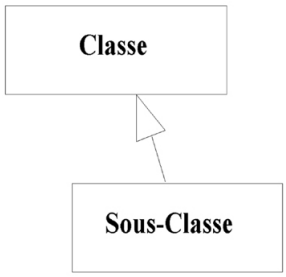
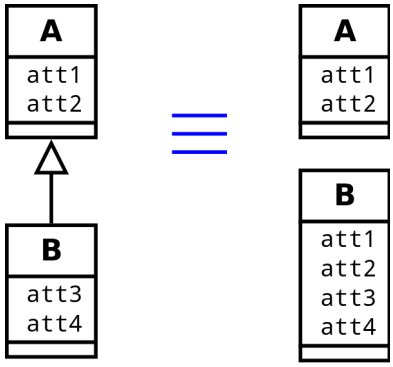


Image 2 Notation de l'héritage en UML

Fondamental: Factorisation

Outre qu'il permet de représenter une relation courante dans le monde réel, l'héritage a un avantage pratique, celui de factoriser la définition de propriétés identiques pour des classes proches.

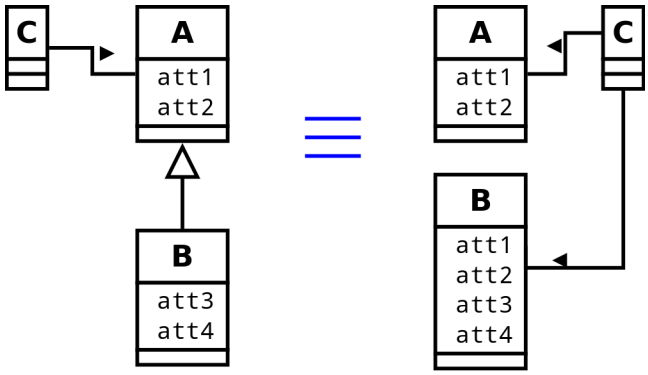


Héritage et factorisation

Fondamental: Is-a

L'héritage permet de représenter la relation "est-un" entre deux objets (*is-a* en anglais).

Donc tout ce qui est vrai pour la classe mère est vrai pour ses classes filles. En particulier si une classe C exprime une association avec une classe A dont hérite B, cela signifie que C peut être associée à B.



Héritage et propriété "is-a"

Exemple : La classe Conducteur

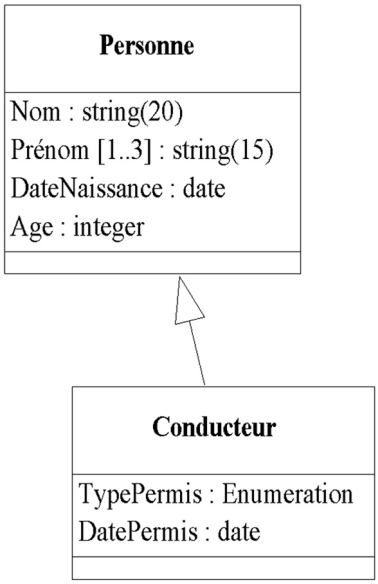


Image 3 Représentation d'héritage en UML

Dans cet exemple la classe Conducteur hérite de la classe Personne, ce qui signifie qu'un objet de la classe conducteur aura les attributs de la classe Conducteur (TypePermis et DatePermis) mais aussi ceux de la classe Personne (Nom, Prénom, DateNaissance et Age). Si la classe Personne avait des méthodes, des associations..., la classe Conducteur en hériterait de la même façon.

c) Exemple : Des voitures et des conducteurs

Exemple

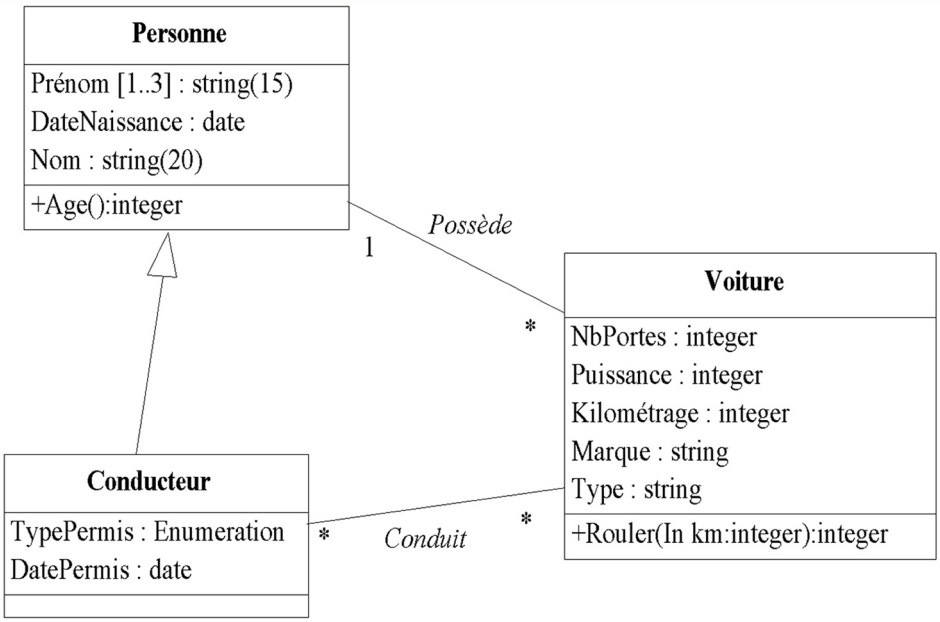


Image 4 Exemple très simple de diagramme de classes

Les relations de ce diagramme expriment que les conducteurs sont des personnes qui ont un permis ; que toute voiture est possédée par une unique personne (qui peut en posséder plusieurs) ; que les voitures peuvent être conduites par des conducteurs et que les conducteurs peuvent conduire plusieurs voitures.

Remarque

Les mots clés in, out et in/out devant un paramètre de méthode permettent de spécifier si le paramètre est une donnée d'entrée, de sortie, ou bien les deux.

Attention

Le but d'une modélisation UML n'est pas de représenter la réalité dans l'absolu, mais plutôt de proposer une vision d'une situation réduite aux éléments nécessaires pour répondre au problème posé. Donc une modélisation s'inscrit toujours dans un contexte, et en cela l'exemple précédent reste limité car son contexte d'application est indéfini.

d) Classe abstraite

Définition : Classe abstraite

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

Attention : Classe abstraite et héritage

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.

Syntaxe : Notation d'une classe abstraite en UML

On note les classes abstraites en italique.

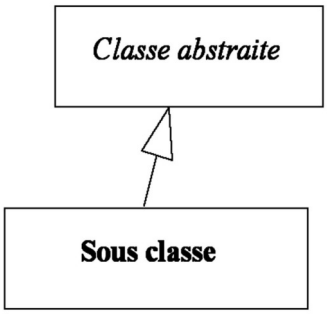


Image 5 Notation d'une classe abstraite en UML

Exemple : Exemple de classes abstraites

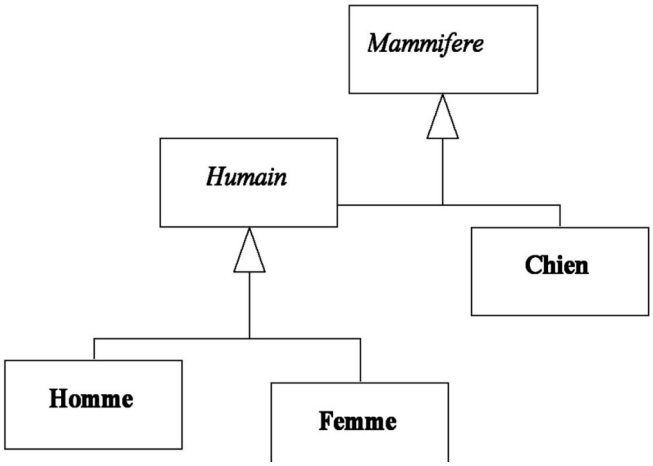


Image 6 Des chiens et des hommes

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.

2. Notions avancées pour l'usage de l'héritage en modélisation des BD

Objectifs

Savoir utiliser l'héritage lors une modélisation

a) Lab II

[15 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**.

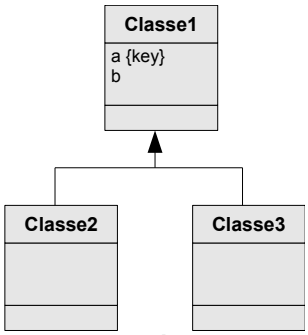
Question

Réaliser le modèle conceptuel de données en UML du problème.

b) Héritage complet

Définition : Héritage complet et presque complet

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



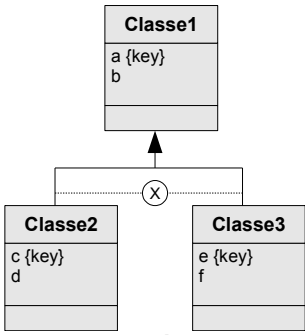
Graphique 1 Héritage complet

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

c) Héritage exclusif

Définition : Héritage exclusif

Un héritage est exclusif si les objets d'une classe fille ne peuvent appartenir aussi à une autre classe fille. On peut le noter avec la contrainte `{X}` sur le diagramme UML (`{XT}` si la classe mère est abstraite).



Graphique 2 Héritage exclusif

Définition : Héritage non exclusif

L'héritage non exclusif est une forme d'héritage qui exprime qu'un objet peut appartenir à deux classes filles en même temps.

Syntaxe : Notation des héritages exclusifs

Normalement la notation UML par défaut désigne un héritage non exclusif.

Mais l'héritage exclusif étant plus commun, on conviendra plutôt qu'un schéma UML qui ne comporte que des héritages non contraints exprime par défaut des héritages exclusifs.

Une note peut être ajoutée précisant que tous les héritages sont exclusifs pour lever toute ambiguïté.

En revanche un schéma qui comporte explicitement des contraintes sur certains héritages est pas sur d'autres permet effectivement d'exprimer de l'héritage non exclusif.

Une dernière possibilité, moins standard, est de marquer d'une contrainte de type {NON X} un héritage non exclusif.

Méthode : On évitera l'héritage non exclusif pour la conception de BD

Bien que cette forme d'héritage soit autorisée en modélisation conceptuelle de bases de données et en UML, on évitera de la mobiliser, sauf en cas d'apport vraiment important d'expressivité, car elle a tendance à complexifier la modalisation, que ce soit au niveau de son interprétation humaine ou de son implémentation en machine.

Il est toujours possible d'exprimer différemment un héritage non exclusif :

- en multipliant les classes filles pour les singulariser,
- et/ou en utilisant la composition pour factoriser les parties communes.

Complément

Exemple de contraintes standardisées sur les associations

d) Héritage multiple

Définition : Héritage multiple

L'héritage multiple est une forme d'héritage qui exprime qu'une classe fille hérite de plusieurs classes mères.

Syntaxe : Notation des héritages multiples

On conseillera d'ajouter une note lors de l'utilisation d'un héritage multiple, expliquant le choix de modélisation et actant qu'il ne s'agit pas d'une erreur.

Méthode : On évitera l'héritage multiple pour la conception de BD

On évitera d'utiliser l'héritage multiple en modélisation de bases de données, sauf en cas d'apport vraiment important d'expressivité.

Il est toujours possible de remplacer un héritage multiple par plusieurs héritages simples.

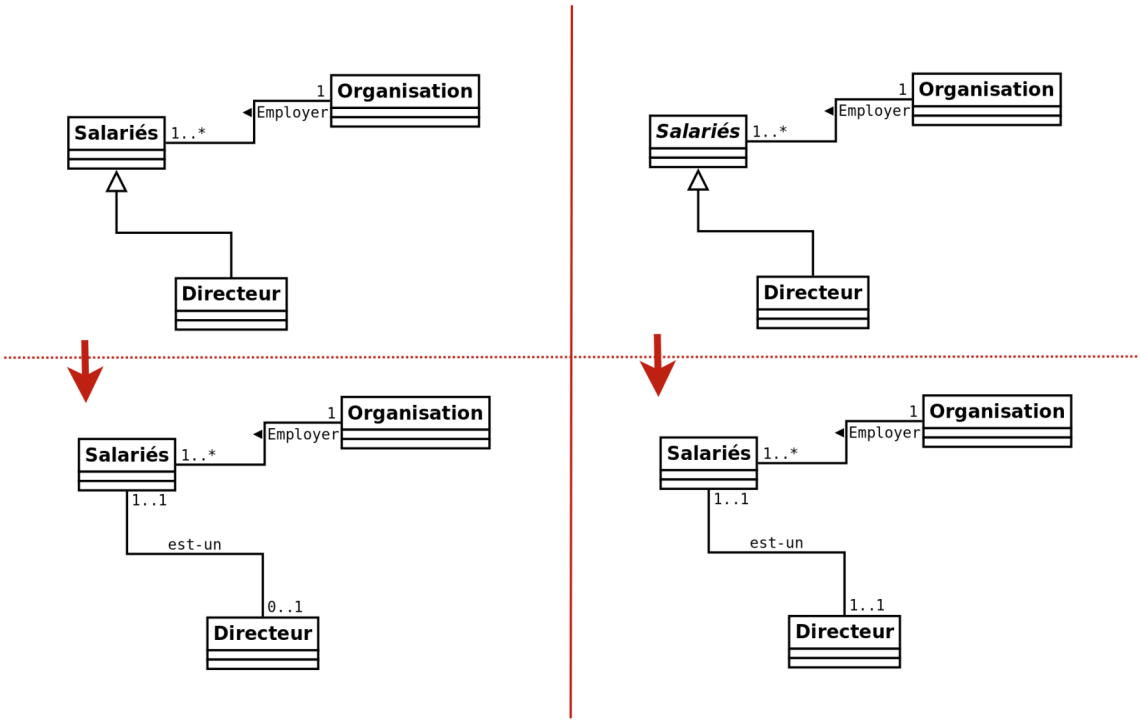
e) Équivalence entre association d'héritage et association 1:1

Remarque

Une association d'héritage est un cas particulier d'association 1:1.

Elle ajoute une sémantique de type "est-un".

Exemple



f) Gestion des défauts

[20 minutes]

Un groupe industriel construisant des moteurs cherche à organiser la gestion des défauts observés sur des moteurs confrontés à des tests en situation réelle. Pour cela un de ses ingénieurs modélise le processus de gestion des défauts, tel qu'il existe actuellement, par le diagramme de classes suivant.

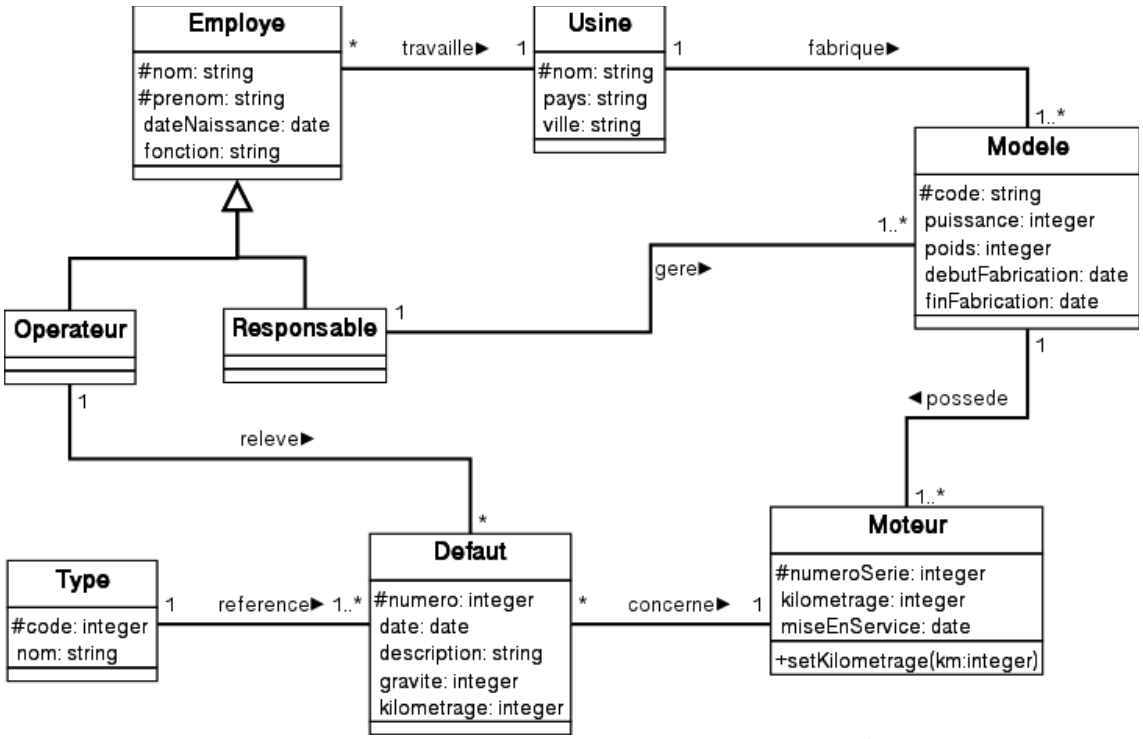


Image 7 Diagramme de classes de gestion des défauts

Question 1

Décrivez en français ce que représente ce diagramme.

Question 2

Étant donné ce modèle, est-il possible de savoir dans quelle usine a été fabriqué un moteur et qui est responsable de sa production ?

Question 3

La responsabilité d'un modèle est elle toujours assumée par un employé travaillant dans l'usine dans laquelle ce modèle est produit ?

Question 4

Pourquoi avoir fait le choix d'une classe **Type** pour codifier les défauts, plutôt qu'un attribut de type énuméré directement dans la classe **Defaut** ?

Question 5

Pourquoi l'attribut `kilométrage` apparaît-il à la fois dans les classes **Defaut** et **Moteur** et pourquoi avoir fait apparaître la méthode `SetKilometrage` ?

Question 6

Ce diagramme permet-il de répondre à la question : Quel est le nombre moyen de défauts rencontrés par les moteurs de chaque modèle ayant été mis en service avant 2000 ? Quelles sont les classes et attributs utiles ?

Question 7

Peut-on également répondre à la question : Quel est le kilométrage moyen des moteurs ayant été concernés par au moins deux défauts d'une gravité supérieure à 5 ?

B. Exercices

1. Capitaine Krik

[30 min]

Le Capitaine Krik a pour tâche de développer une base de données sur les vaisseaux spatiaux et les équipages de la TarFleet.

La flotte ne possède que trois types de vaisseaux : les croiseurs, les frégates et les chasseurs. Chaque membre de la TarFleet a un nom, un prénom, une date de naissance, une planète d'origine et un numéro d'identifiant unique. Certains membres sont aussi soit pilotes, soit capitaines : les pilotes ont un nombre de chasseurs ennemis abattus, tandis que les capitaines ont un certain nombre d'étoiles (entre zéro et cinq).

Krik veut savoir dans la base de données quelles personnes sont affectées à quel vaisseau. Dans la flotte, un chasseur a pour équipage un unique pilote, une frégate a deux pilotes et cinq autres membres d'équipage, tandis qu'un croiseur a un capitaine et de nombreux autres personnels. Les croiseurs peuvent aussi transporter dans leur hangar plusieurs chasseurs (leurs pilotes ne sont alors pas comptés dans l'équipage du croiseur).

Pour chaque vaisseau, on veut connaître son nom et son identifiant, sachant que celui-ci est généré automatiquement à partir du nom et du type de vaisseau (par exemple "EnterpriseCruser"). Pour les frégates et les croiseurs, on veut également connaître la puissance de leur bouclier (les chasseurs n'en sont pas équipés), et, pour un croiseur, le nombre de membres d'équipage maximal qu'il peut accueillir.

Krik veut aussi des informations sur les réacteurs de chaque vaisseau. Les réacteurs sont soit à fission nucléaire, soit à trou noir miniature. Chacun a un numéro d'emplacement (qui indique où le moteur est monté sur le vaisseau), un poids et une poussée, mais les réacteurs à fission ont également une quantité maximale de carburant, tandis que les réacteurs à trou noir ont une puissance critique.

Question

Proposer un schéma UML permettant de modéliser une telle base de données.

2. Armoires secrètes

[20 minutes]

Dans le cadre de la réalisation d'une base de données pour les services secrets français, vous disposez de l'analyse de besoins suivant :

- les agents secrets sont identifiés par un code sur 3 chiffres (comme *007*) et possède un nom et un prénom ;
- les agents secrets produisent des rapports, parfois seul, parfois à plusieurs. Tous les agents secrets ont produit au moins un rapport ;
- les agents secrets sont communément appelés par leurs initiales et leur code : ainsi James Bond 007 est en général appelé *JB007* ;
- un rapport est identifié par un titre (il n'existe pas deux rapports avec le même titre) et il possède une description ainsi que des mots-clés (au moins 2, au plus 10) ;

- le rangement des rapports est organisé comme suit : les rapports sont situés dans des dossiers, qui sont classés dans des casiers, qui sont rangés sur des étagères, dans des armoires ;
- les dossiers, casiers, étagères et armoires sont des rangements, qui sont identifiés par une lettre et un nombre (inférieur à 100). Chaque rangement a une capacité qui détermine le nombre de rangements qu'il peut contenir ;
- il n'existe pas de rapport ou de rangement qui ne serait rangé nulle part.

Question

Proposez un MCD✯ en UML✯ de ce problème. L'on cherchera le modèle le plus expressif possible.

On fera apparaître les types des attributs, en étant le plus précis possible avec les informations dont nous disposons, ainsi que les clés.

Transformation de l'héritage en relationnel

II

A. Cours

1. Les trois représentations de l'héritage en relationnel

Objectifs

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

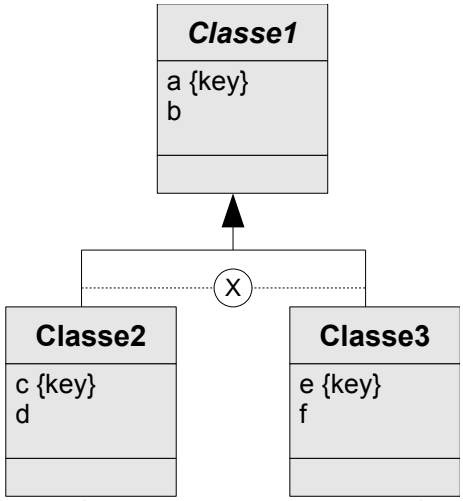
Connaître les choix possibles pour le cas de l'héritage et savoir faire les bons choix.

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.

a) Exercice

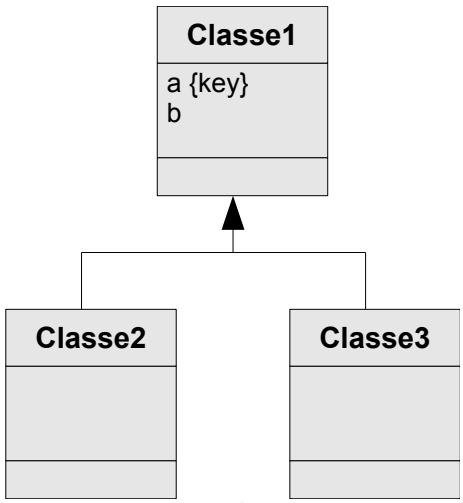
Traduire chacun des schémas conceptuels suivants en schéma relationnel.

Question 1



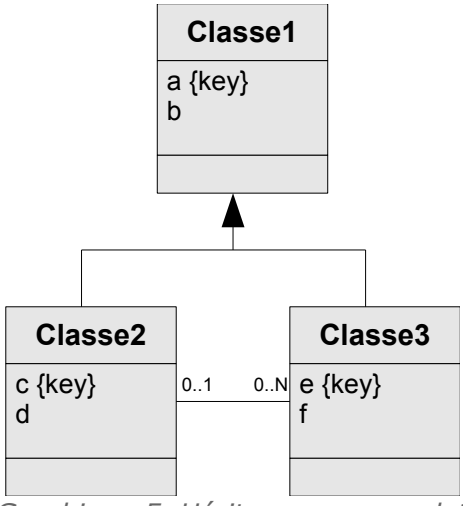
Graphique 3 Héritage exclusif (classe mère abstraite)

Question 2



Graphique 4 Héritage complet

Question 3



Graphique 5 Héritage non complet

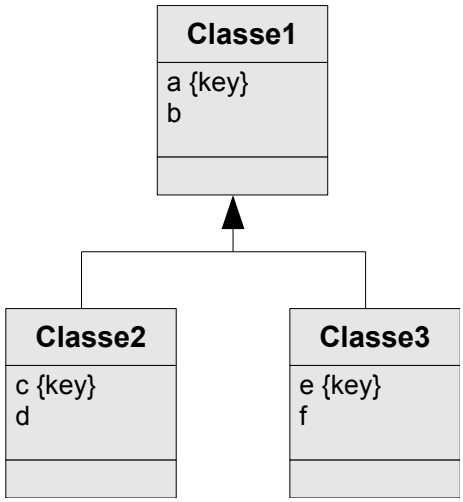
b) Transformation de la relation d'héritage

Fondamental

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.

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.**



Graphique 6 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

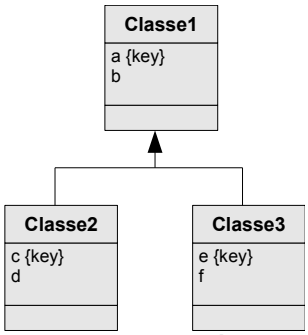
Classe abstraite

Héritage complet

c) Transformation de la relation d'héritage par référence

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.



Graphique 7 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 primaire 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.

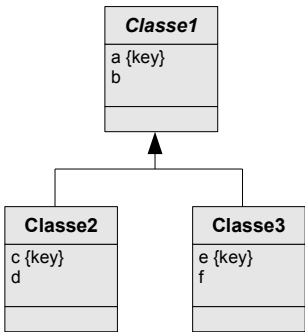
Complément

Héritage par une référence et vues

d) 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.



Graphique 8 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).

Complément : Héritage exclusif

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

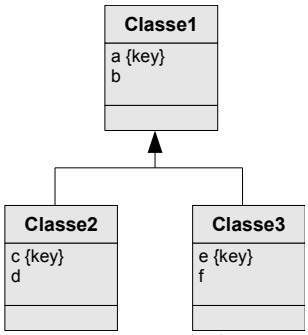
Héritage par les classes filles et vues

e) 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.



Graphique 9 Héritage

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

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).

Conseil : Classe mère non abstraite

Cette solution est particulièrement adaptée lorsque la classe mère n'est pas abstraite, car cela en autorise l'instanciation naturellement. Lorsque la classe mère est abstraite il est moins naturel de disposer d'une table associée à cette classe.

Conseil : Héritage complet

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, a fortiori si la classe mère n'est pas abstraite.

Complément : Héritage non exclusif

afin de gérer l'héritage non exclusif (un objet peut être de plusieurs classes filles à la fois), le domaine de l'attribut de discrimination, peut être étendu à des combinaisons de noms des différentes classes filles.

```
Classe1(#a,b,c,d,e,f,t:{1,2,3,23})
```

Complément : Représentation de la discrimination par des booléens

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.

Dans cette configuration la classe mère sera logiquement considérée comme non abstraite et un objet appartiendra à la classe mère si tous ses attributs de discrimination valent "faux". Seule une contrainte dynamique permettra de définir la classe mère comme abstraite, en précisant que les attributs de discrimination ne peuvent être tous à "faux".

```
Classe1(#a,b,c,d,e,f,t:{1,2,3,23})
```

équivaut à :

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

Complément

Héritage par la classe mère et vues

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

a) Éléments de choix

Méthode : Choisir le bon mode de transformation d'une relation d'héritage

La difficulté consiste donc pour chaque relation d'héritage à choisir le bon mode de transformation, sachant que chaque solution possède ses avantages et ses inconvénients.

	Inconvénients	Cas d'usage
Par référence	Lourdeur liée à la nécessité de représenter les données des classes filles sur deux relations	Adapté à tous les cas, particulièrement lorsque la classe mère n'est pas abstraite
Par les classes filles	Les associations avec la classe mère peuvent être problématiques ; redondance dans le cas de l'héritage non exclusif	Adapté à l'héritage exclusif, particulièrement lorsque la classe mère est abstraite et ne comporte pas d'association
Par la classe mère	Nullité systématique pour les attributs des classes filles (et pour la classe mère si celle-ci n'est pas abstraite) ; héritage non exclusif et non complet problématique	Adapté à l'héritage complet et presque complet, particulièrement lorsque la classe mère n'est pas abstraite

	Classe mère abstraite	Classe mère non abstraite	Héritage exclusif	Héritage non exclusif	Héritage complet	Héritage presque complet	Héritage non complet	Association classe mère
Par référence								
Par les classes filles								
Par la classe mère								

Tableau 1 Choix de la bonne transformation

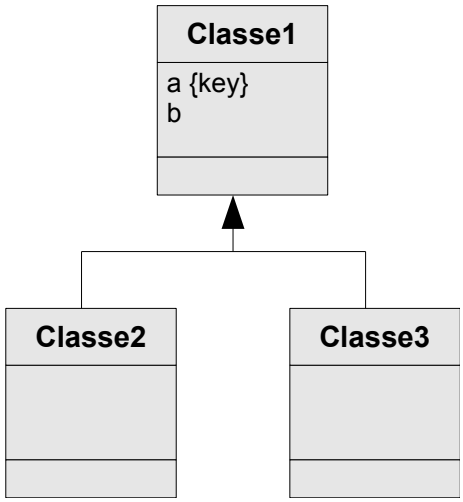
Complément

Classes abstraites
Héritage complet
Héritage exclusif

b) Cas simples

Méthode : Héritage complet

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

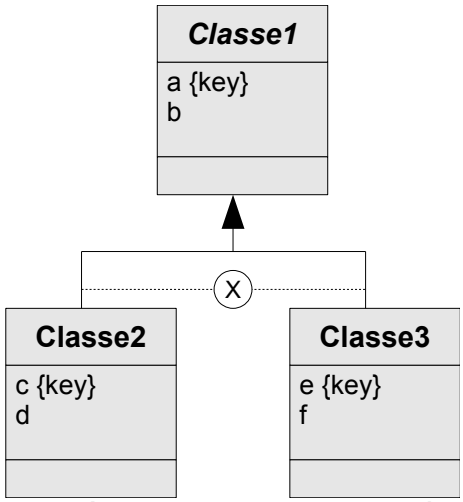


Graphique 10 Héritage complet

1. Si la classe mère est abstraite :
   `Classe1(#a,b,t:{2,3})`
2. Si la classe mère n'est pas abstraite :
   `Classe1(#a,b,t:{1,2,3})`

Méthode : Héritage non complet avec classe mère abstraite et sans association

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

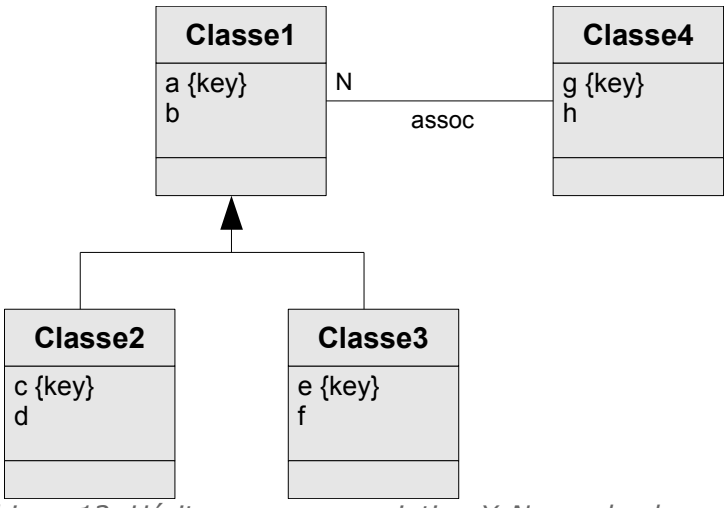


Graphique 11 Héritage exclusif (classe mère abstraite)

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

Méthode : Héritage presque complet

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.

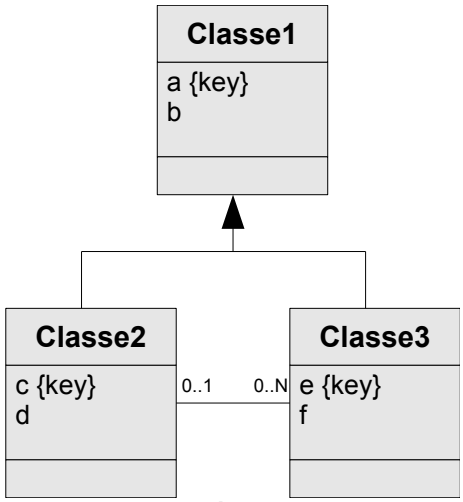


Graphique 12 Héritage avec association X:N vers la classe mère

```
Classe1(#a,b,c,d,e,f,t:{1,2,3})
Classe4(#g,h,fka=>Classe1)
```

Méthode : Héritage non complet avec classe mère non abstraite et sans association

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

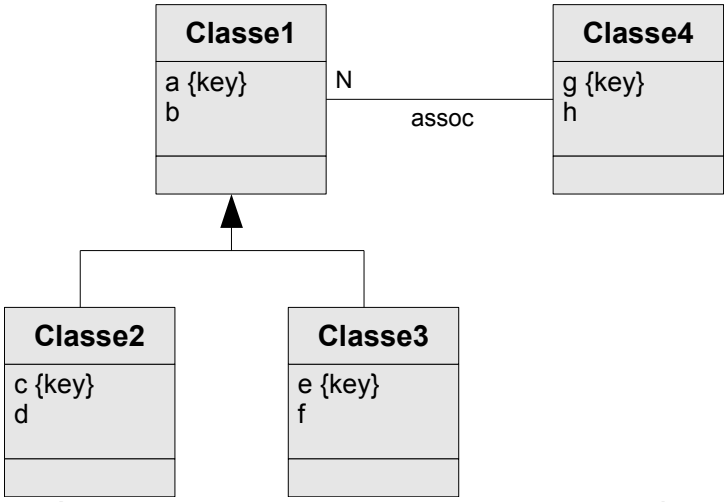


Graphique 13 Héritage non complet

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

c) Cas problématiques

Attention : Héritage par les classes filles avec association X:N vers la classe mère



Héritage avec association X:N vers la classe mère

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

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

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

Contraintes : `fka OR fkb`

La seule solution, peu élégante, 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. Cette solution devient ingérable si le nombre de classes filles augmente.

Attention : Héritage non complet par la classe mère

Héritage non complet

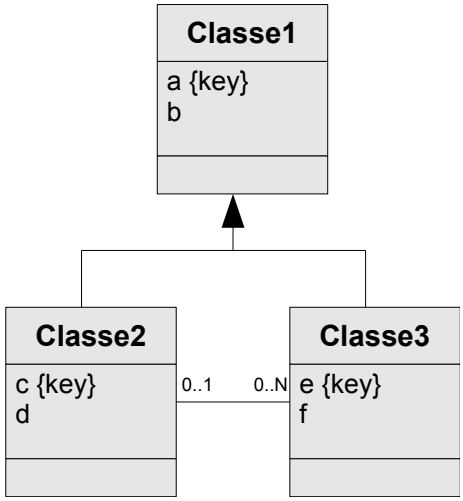
Classe1(#a,b,c,d,e,f,t:{1,2,3})

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

Contraintes : fka ne référence que des enregistrements tels que 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.

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



Héritage non complet

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

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

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).

Complément : Héritage non exclusif

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

Complément : Héritage multiple

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

Complément

Héritage complet

Classes abstraites

d) Exemple de transformation d'une relation d'héritage

Exemple

Soit le modèle UML suivant :

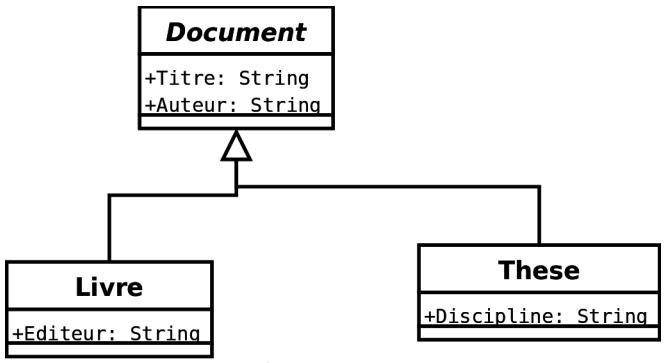


Image 8 Représentation de documents

Il existe trois façons de traduire la relation d'héritage : par référence, par absorption par les classes filles, par absorption par la classe mère.

Remarque : Type d'héritage

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.

La meilleure solution est donc ici un **héritage par les classes filles**, mais un héritage par la classe mère reste un bon choix. En revanche l'héritage par référence est un mauvais choix, car il conduit à une solution inutilement complexe.

Observons néanmoins les avantages et inconvénients que chacun des trois choix porterait.

i Héritage représenté par une référence

```
1  Document(#Titre:Chaîne, Auteur:Chaîne) avec Document AND (These OR Livre)
2  These(#Titre=>Document, Discipline:Chaîne)
3  Livre(#Titre=>Document, Editeur:Chaîne)
4
```

Complément

```
1
2  vThese = Jointure (Document, These, Document.Titre=These.Titre)
3  vLivre = Jointure (Document, Livre, Document.Titre=Livre.Titre)
```

```
4 Contraintes : PROJ(Document,Titre) IN (PROJ(These,Titre) UNION PROJ(Livre,Titre))
```

Remarque : Avantages du choix

Il n'y a ni redondance ni valeur nulle inutile dans les relations These et Livre.

Remarque : Inconvénients du choix

- Il est relativement lourd de devoir créer un tuple dans Document pour chaque tuple dans These ou Livre, alors que la relation Document ce porte que l'information concernant l'auteur, juste pour assurer les cas, par ailleurs plutôt rare dans la réalité, où les thèses sont publiées sous forme de livres.
- De plus la classe Document étant abstraite, il ne devra jamais y avoir de tuple dans document qui n'a pas de tuple correspondant dans Livre ou These. Ceci n'est pas contrôlable directement en relationnel et devra donc être vérifié au niveau applicatif (ou bien la classe ne devra plus être considérée comme abstraite).

ii Héritage absorbé par les classes filles

```
1 These(#Titre, Discipline:Chaîne, Auteur:Chaîne)
2 Livre(#Titre, Editeur:Chaîne, Auteur:Chaîne)
```

Remarque : Avantages du choix

Il est plus simple que le précédent, puisque la représentation d'une thèse ou d'un livre ne nécessite plus que d'ajouter un seul tuple. Il évite donc les clés étrangères, toujours plus délicates à gérer d'un point de vue applicatif.

Remarque : Inconvénient du choix

Si l'héritage est exclusif ce modèle n'a aucun inconvénient.

Si l'héritage n'est pas exclusif et qu'une thèse est aussi un livre, alors une information sera dupliquée (l'auteur), ce qui introduit de la redondance.

iii Héritage absorbé par la classe mère

```
1 Document(#Titre, Discipline:Chaîne, Editeur:Chaîne, Auteur:Chaîne, Type:{These|
  Livre})
2
```

Complément

```
1
2 vThese = Projection (Restriction (Document, Type='These'), Titre, Discipline,
  Auteur)
3 vLivre = Projection (Restriction (Document, Type='Livre'), Titre, Editeur,
  Auteur)
```

Remarque : Avantages du choix

Il est plus simple que le premier, puisque la représentation d'une thèse ou d'un livre ne nécessite plus que d'ajouter un seul tuple. Il évite donc les clés étrangères..

Remarque : Inconvénient du choix

- Les tuples de Document contiendront systématiquement une valeur nulle soit pour l'éditeur soit pour la discipline.
- Il faut vérifier l'adéquation entre la valeur de Type et le fait que Discipline ou Auteur n'est pas valué.

- Si l'héritage est exclusif (ce qui est le cas par défaut), alors les tuples de la relation Document contiendront systématiquement une valeur nulle soit pour l'éditeur soit pour la discipline.
- Il faudra pour être rigoureux vérifier l'adéquation entre la valeur de Type et le fait que Discipline ou Auteur n'est pas valué (si Type='These' alors Editeur=Null et si Type='Livre' alors Discipline=Null).

B. Exercices

1. Lab II+

[30 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. 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 :

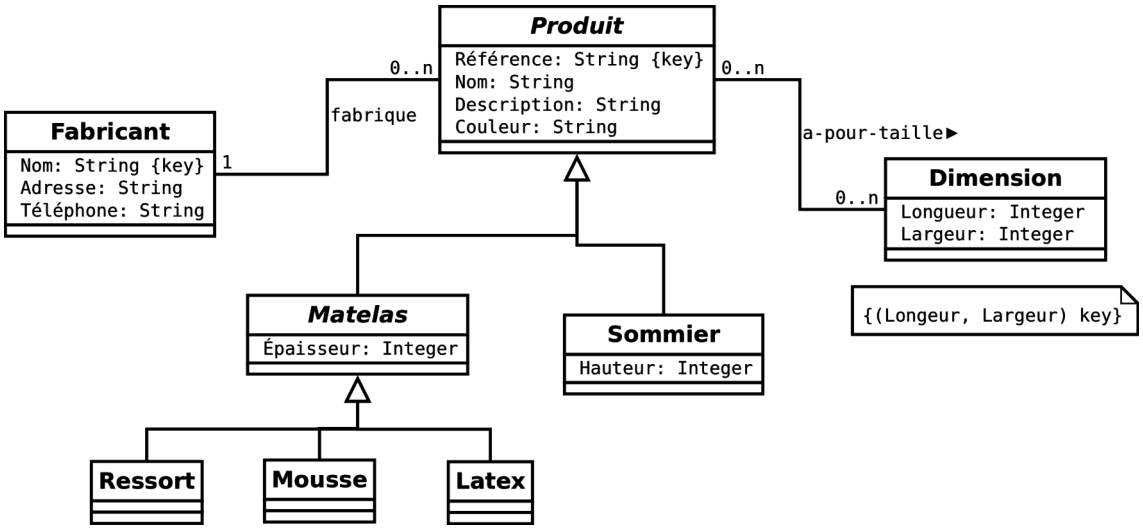
« Le fonctionnement actuel est relativement simple, nous souhaitons juste l'automatiser pour éviter les erreurs et les pertes de temps. Nous avons des types de produits, qui sont des matelas ou des sommiers. Chaque type de produit est caractérisé par des références uniques que nous créons en plus du 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

3. 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.

III

Modélisation avancée des associations en UML et en relationnel

A. Cours

1. Modélisation avancée des associations en UML

Objectifs

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

a) Exercice : Entreprise

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

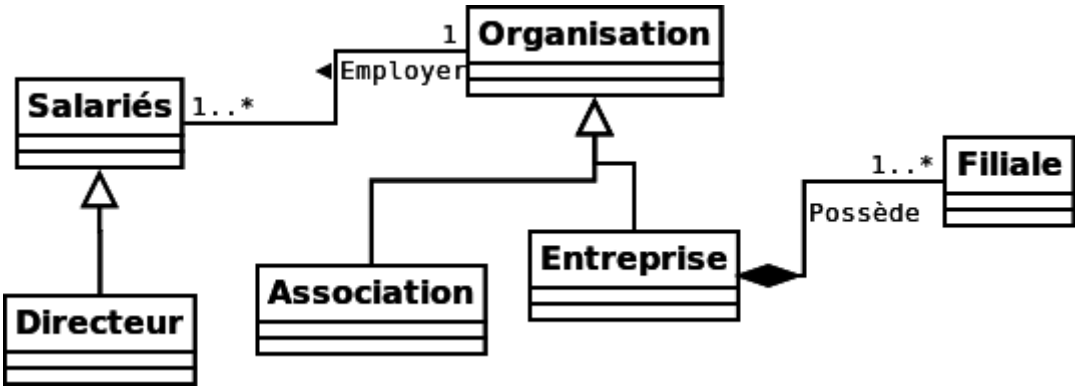


Image 9 MCD UML

☐	Une association peut employer un directeur.
☐	Une association peut employer plusieurs directeurs.
☐	Une association peut ne pas employer de directeur.
☐	Une filiale peut appartenir à plusieurs entreprises.
☐	Il existe des organisations qui ne sont ni des entreprises ni des associations.

b) Composition

Définition : Association de composition

On appelle composition une association particulière qui possède les propriétés suivantes :

- La composition associe une classe composite et des classes parties, tel que tout objet partie appartient à un et un seul objet composite. C'est donc une association 1:N (voire 1:1).
- La composition n'est pas partageable, donc un objet partie ne peut appartenir qu'à un seul objet composite à la fois.
- Le cycle de vie des objets parties est lié à celui de l'objet composite, donc un objet partie disparaît quand l'objet composite auquel il est associé disparaît.

Remarque

- La composition est une association particulière (binaire de cardinalité contrainte).
- La composition n'est pas symétrique, une classe joue le rôle de conteneur pour les classes liées, elle prend donc un rôle particulier a priori.
- La composition est une agrégation avec des contraintes supplémentaires (non partageabilité et cycle de vie lié).

Syntaxe : Notation d'une composition en UML

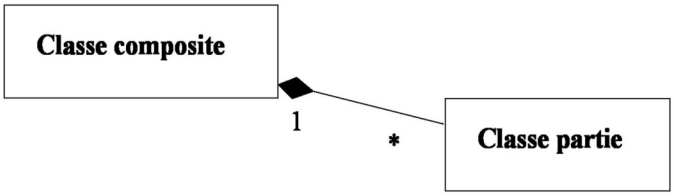


Image 10 Notation de la composition en UML

Attention : Composition et cardinalité

La cardinalité côté composite est toujours de exactement 1.
Côté partie la cardinalité est libre, elle peut être 0..1, 1, * ou bien 1..*.

Exemple : Exemple de composition

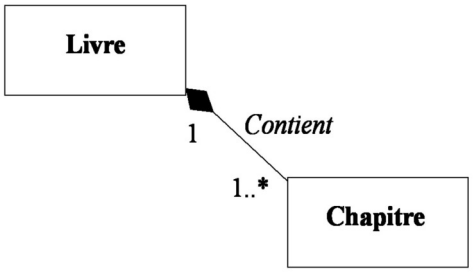


Image 11 Un livre

On voit bien ici qu'un chapitre n'a de sens que faisant partie d'un livre, qu'il ne peut exister dans deux livres différents et que si le livre n'existe plus, les chapitres le composant non plus.

Remarque : Composition et entités faibles

La composition permet d'exprimer une association analogue à celle qui relie une entité faible à une entité identifiante en modélisation E-A✯. L'entité de type faible correspond à un objet partie et l'entité identifiante à un objet composite.

Conseil : Composition et attribut multivalué

- Une composition avec une classe partie dotée d'un seul attribut peut s'écrire avec un attribut multivalué.
- Un attribut composé et multivalué peut s'écrire avec une composition.

Rappel : Voir aussi

- *Attributs*
- *Agrégation*

c) Agrégation

Définition : Association d'agrégation

L'agrégation est une association particulière utilisée pour préciser une relation tout/partie (ou ensemble/élément), on parle d'association **méréologique**.

Elle possède la propriété suivante : L'agrégation associe une classe agrégat et des classes parties, tel que tout objet partie appartient à au moins un objet agrégat.

Remarque

- L'agrégation est une association particulière (binaire de cardinalité libre).
- L'agrégation n'est pas symétrique.

Syntaxe

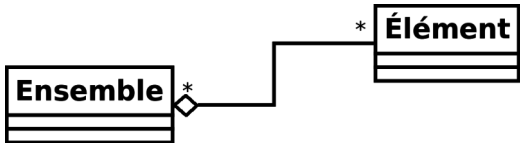


Image 12 Notation de l'agrégation en UML

La cardinalité, peut être exprimée librement, en particulier les instances de la classe Élément peuvent être associées à plusieurs instances de la classe Ensemble, et même de plusieurs classes.

Attention

L'agrégation garde toutes les propriétés d'une association classique (cardinalité, cycle de vie, etc.), elle ajoute simplement une terminologie un plus précise via la notion de tout/partie.

d) Explicitation des associations

Syntaxe : Sens de lecture

Il est possible d'ajouter le sens de lecture du verbe caractérisant l'association sur un diagramme de classe UML, afin d'en faciliter la lecture. On ajoute pour cela un signe < ou > (ou un triangle noir) à côté du nom de l'association

Syntaxe : Rôle

Il est possible de préciser le rôle joué par une ou plusieurs des classes composant une association afin d'en faciliter la compréhension. On ajoute pour cela ce rôle à côté de la classe concernée (parfois dans un petit encadré collé au trait de l'association.

Exemple

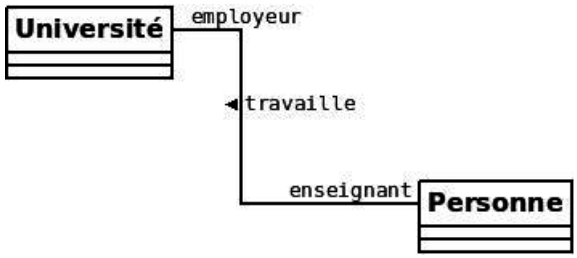


Image 13 Rôle et sens de lecture sur une association

Définition : Association réflexive

Une association réflexive est une association qui associe une classe avec elle-même.

L'explicitation des associations est particulièrement utile dans le cas des associations réflexives.

e) Classe d'association avec clé locale

Rappel

Classe d'association

Contrainte inhérente à la relation N:M

Dans l'exemple suivant, chaque personne peut avoir un emploi dans plusieurs sociétés, mais elle ne peut pas avoir plusieurs emplois dans une même société.

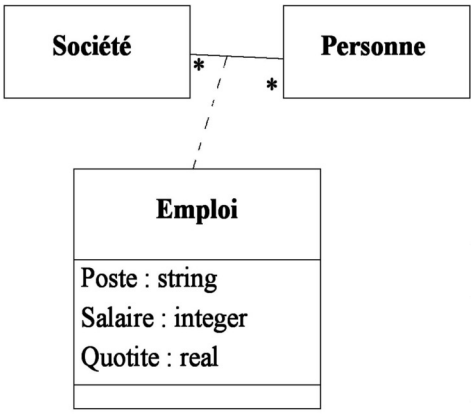


Image 14 Emplois

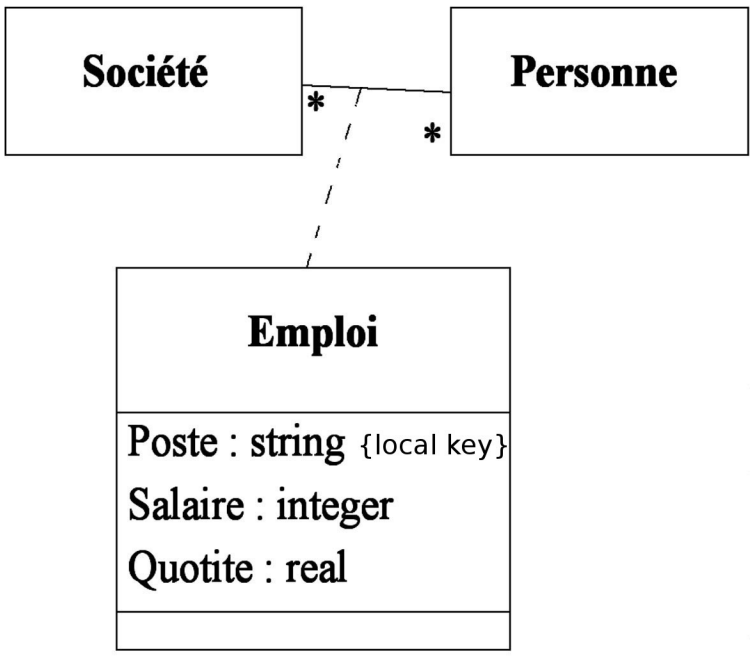
La transformation en relationnelle est cohérente avec cette contrainte.

```
1  Société(...)
2  Personne(...)
3  Emploi(#personne=>Personne, #societe=>Societe, poste:string, salaire:integer,
   quotite:numeric(1,2))
```

#personne	#societe	poste	salaire	quotite
~~Al~~	~~Canonical~~	Directeur	100000	0,5
~~Al~~	~~Canonical~~	Développeur	50000	0,5

Méthode : Intérêt de la clé locale

La spécification d'une clé locale dans emploi, par exemple ici le poste, permet de lever cette contrainte, lorsqu'on le souhaite, en permettant d'identifier chaque instance de l'association, ici l'emploi d'une personne par sa société.



La transformation en relationnelle permettra de maintenir la modélisation, en ajoutant la clé locale à la clé initiale

```
1  Société(...)
2  Personne(...)
3  Emploi(#personne=>Personne, #societe=>Societe, #poste:string, salaire:integer,
   quotite:numeric(7,2))
```

#personne	#societe	#poste	salaire	quotite
Al	Canonical	Directeur	100000	0,5
Al	Canonical	Développeur	50000	0,5

f) Associations ternaires

Syntaxe

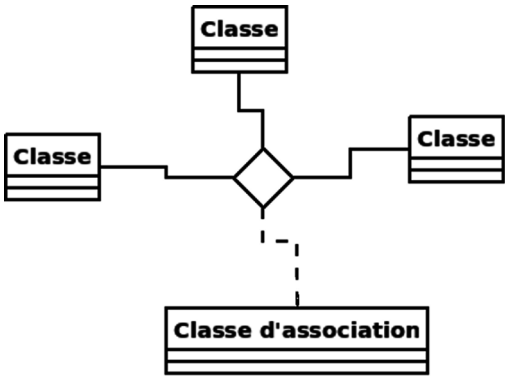


Image 15 Notation d'une association ternaire

Conseil : Ne pas abuser des associations ternaires

Il est toujours possible de réécrire une association ternaire avec trois associations binaires, en transformant l'association en classe.

Conseil : Pas de degré supérieur à 3

En pratique on n'utilise jamais en UML d'association de degré supérieur à 3.

2. Passage UML-Relationnel : Associations avancées

Objectifs

Savoir faire le passage d'un schéma conceptuel UML à un schéma relationnel dans tous les cas.

Reconnaître les cas de transformation qui se traitent toujours de la même façon et ceux qui nécessite une modélisation complémentaire.

a) Composition et agrégation

Question

Traduire chacun des schémas conceptuels suivants en schéma relationnel.

Graphique 14 Composition

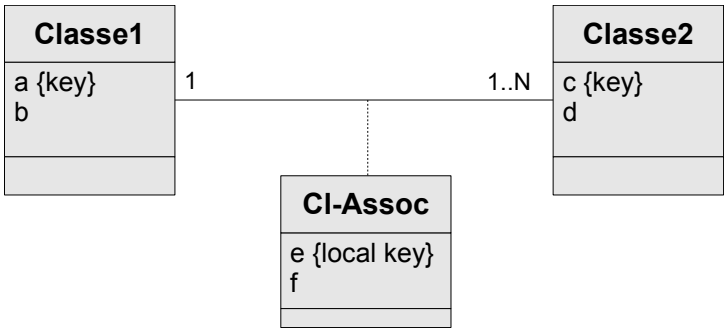
Graphique 15 Agrégation 1:N

Graphique 16 Agrégation N:M

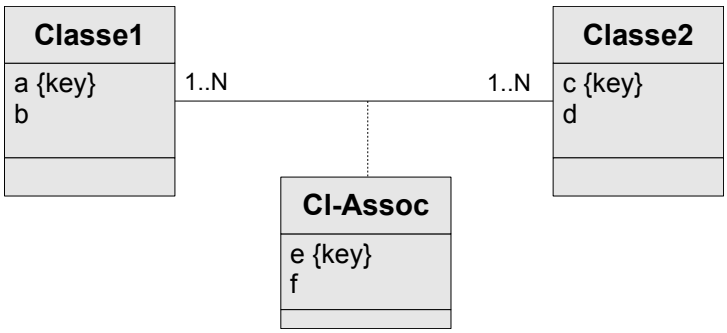
b) Classe d'association

Question

Traduire chacun des schémas conceptuels suivants en schéma relationnel.



Graphique 17 Classe d'association (1:N)



Graphique 18 Classe assocation (N:M)

c) Transformation des compositions

Méthode

Une composition

- est transformée comme une association 1:N,
- puis on ajoute à la clé de la classe partie (dite clé locale) la clé étrangère vers la classe composite pour construire une clé primaire composée.

Graphique 19 Composition

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

Remarque : Clé locale

Pour identifier une classe partie dans une composition, on utilise une clé locale concaténée à la clé étrangère vers la classe composite, afin d'exprimer la dépendance entre les deux classes.

Si une clé naturelle globale permet d'identifier de façon unique une partie indépendamment du tout, on préférera la conserver comme clé candidate plutôt que de la prendre pour clé primaire.

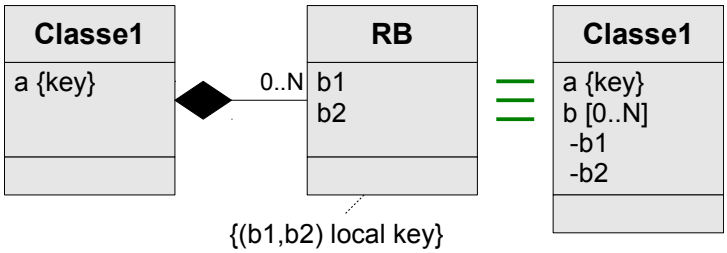
Si on la choisit comme clé primaire cela revient à avoir transformé la composition en agrégation, en redonnant une vie propre aux objets composants.

Complément : Composition et entités faibles en E-A

Une composition est transformée selon les mêmes principes qu'une entité faible en E-A.

Complément : Attributs multivalués et composés

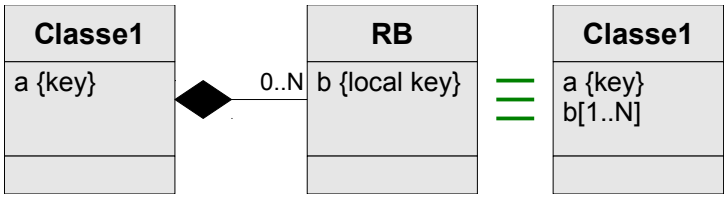
La transformation d'un attribut composé multivalué donne un résultat équivalent à la transformation d'une composition.



Graphique 20 Composition et attribut composé multivalué

```
Classe1(#a)
RB(#b_b1,#b_b2,#a=>Classe1)
```

La transformation d'une composition avec un seul attribut pour la classe composante donne un résultat équivalent à la transformation d'un attribut multivalué.



Graphique 21 Composition et attribut multivalué

```
Classe1(#a)
RB(#b,#a=>Classe1)
```

Rappel : Voir aussi

Transformation des attributs

d) Transformation des agrégations

Rappel : Agrégation

Les associations de type agrégation se traitent de la même façon que les associations classiques.

Graphique 22 Agrégation 1:N

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

Graphique 23 Agrégation N:M

```
Classe1(#a,b)
```

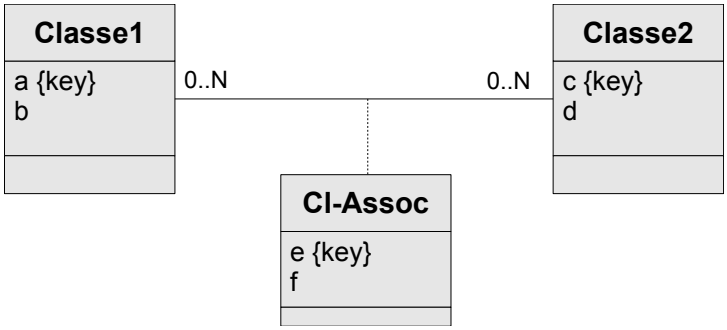
```
Classe2(#c,d)
Assoc(#a=>Classe1,#c=>Classe2)
```

e) Transformation des classes d'association avec clé locale

Méthode : Classe d'association N:M

Les attributs de la classe d'association,

- sont ajoutés à la relation issue de l'association N:M ;
- la clé locale de la classe d'association est concaténée aux clés étrangères composant déjà la clé primaire de la relation d'association.



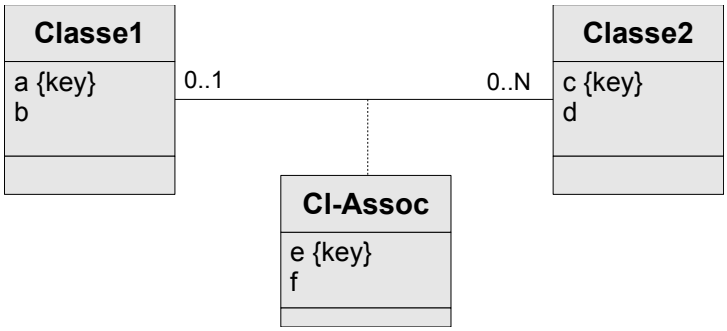
Graphique 24 Classe assocation (N:M)

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

Méthode : Classe d'association 1:N

Les attributs de la classe d'association,

- sont ajoutés à la relation issue de la classe côté N ;
- la clé locale devient un clé candidate.



Graphique 25 Classe d'association (1:N)

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

f) Correspondance entre UML et relationnel

Modèle UML	Modèle relationnel
Classe instanciable	Relation
Classe abstraite	Rien
Objet	Nuplet
Attribut simple	Attribut atomique
Attribut composite	Ensemble d'attributs
Attribut multivalué	Relation et clé étrangère
Attribut dérivé	Procédure stockée ou contrainte dynamique
Méthode	Procédure stockée
Association 1:N	Clé étrangère
Association N:M	Relation et clé étrangère
Association de degré 3 ou supérieur	Relation et clé étrangère
Classe d'association	Attributs
Composition	Clé

Tableau 2 Passsage UML vers Relationnel

g) Exercice

Quel(s) schéma(s) relationnel(s) corresponde(nt) au modèle UML suivant ?

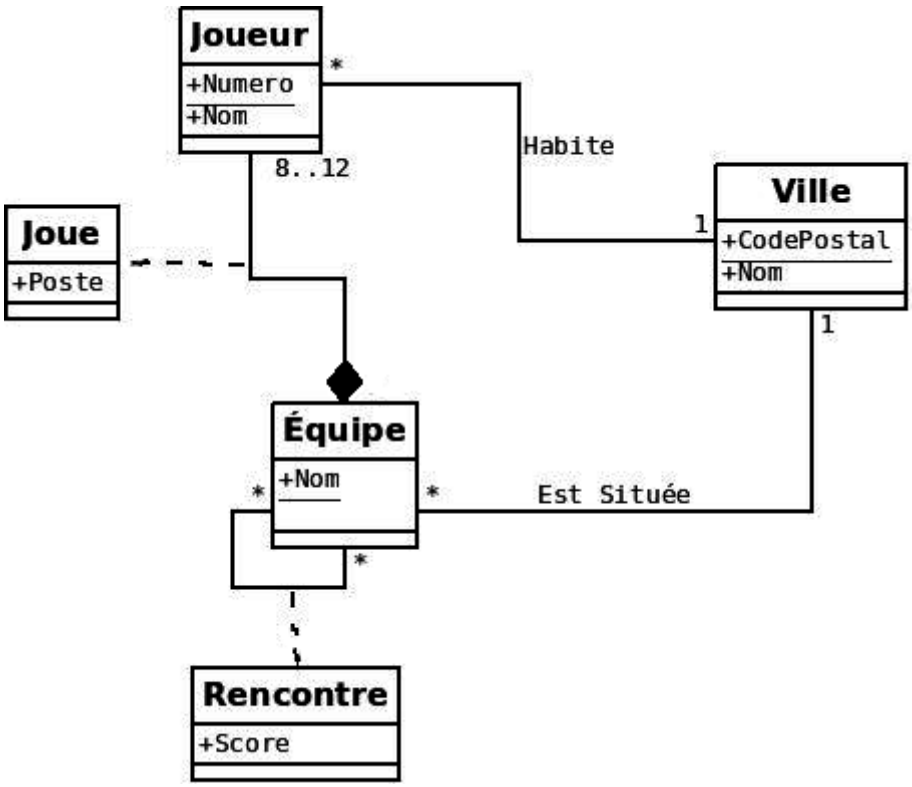


Image 16 Volley ball

☐	Joueur(#Numero, Nom, Ville=>Ville) Equipe(#Nom, Joueur=>Joueur, Poste, Rencontre=>Equipe, Score, Ville=>Ville) Vile(#CodePostal, Nom)

☐	Joueur(#Numero, #Equipe=>Equipe, Nom, Poste, Ville=>Ville)
	Equipe(#Nom, Ville=>Ville)
	Ville(#CodePostal, Nom)
	Rencontre(#Equipe1=>Equipe, #Equipe2=>Equipe, Score)

☐	Joueur(#Numero, Nom, Ville=>Ville) Equipe(#Nom, Joueur=>Joueur, Poste, Ville=>Ville) Ville(#CodePostal, Nom) Rencontre(#Equipe1=>Equipe, #Equipe2=>Equipe, Score)

☐	Joueur(#Numero, Nom)
	Equipe(#Nom)
	Ville(#CodePostal, Nom)
	Habite(#Numero=>Joueur, #CodePostal=>Ville)
	Joue(#Numero=>Joueur, #Nom=>Equipe, Poste)
	EstSituee(#Nom=>Equipe, #CodePostal=>Ville)
	Rencontre(#NomE1=>Equipe, #NomE2=>Equipe, Score)

h) Exercice

Soit le schéma UML suivant :

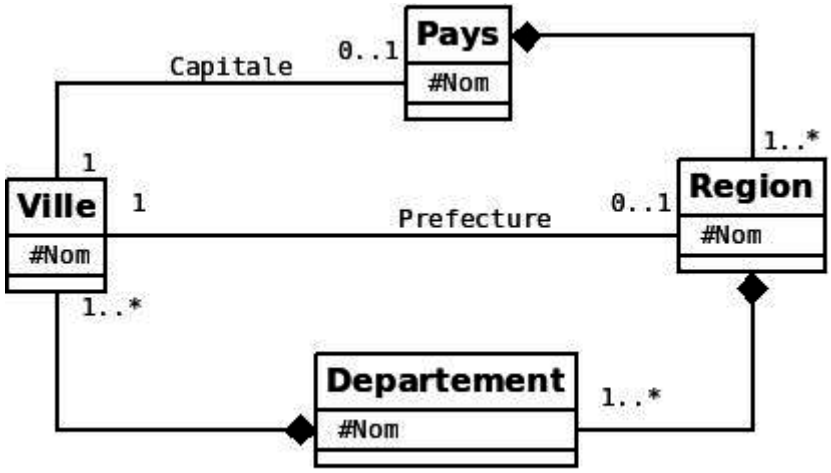


Image 17 Villes

Quelles sont les modèles relationnels qui correspondent à ce modèle conceptuel ?

☐	Pays(#Nom, Capitale=>Ville) Region(#Nom, Prefecture=>Ville, Pays=>Pays) Departement(#Nom, Region=>Region) Ville(#Nom, Departement=>Departement)
☐	Pays(#Nom, Capitale=>Ville) Region(#Nom, Prefecture=>Ville, Pays=>Pays) Departement(#Nom, Region=>Region, Pays=>Pays) Ville(#Nom, Departement=>Departement, Region=>Region, Pays=>Pays)
☐	Pays(#Nom) Region(#Nom, Pays=>Pays) Departement(#Nom, Region=>Region) Ville(#Nom, Departement=>Departement, Capitale=>Pays, Prefecture=>Region)

☐	Pays(#Nom, Capitale=>Ville)
	Region(#Nom, #Pays=>Pays, Prefecture=>Ville)
	Departement(#Nom, #Region=>Region)
	Ville(#Nom, #Departement=>Departement)

☐	Pays(#Nom, CapitaleVille=>Ville, CapitaleDepartement=>Ville, CapitaleRegion=>Ville, CapitalePays=>Ville)
	Region(#Nom, #Pays=>Pays, PrefectureVille=>Ville, PrefectureDepartement=>Ville, PrefectureRegion=>Ville, PrefecturePays=>Ville)
	Departement(#Nom, #Region=>Region, #Pays=>Region)
	Ville(#Nom, #Departement=>Departement, #Region=>Departement, #Pays=>Departement)

3. Synthèse sur la modalisation UML et relationnelle

a) Quelques éléments de stylistique UML

Attention

- **Toutes les associations doivent être nommées (sauf composition, héritage, agrégation).**
- **Ne pas utiliser de nom générique pour les Classes comme "Entité", "Classe", "Objet", "Truc"...**
- **Éviter les noms génériques pour les associations (comme "est associé à")**
- **Attention au sens des compositions et agrégation, le losange est côté ensemble, et n'oubliez pas les cardinalités, notamment côté parties.**

Conseil

- N'utilisez pas le souligné ni les # en UML pour identifier les clés, préférez la contrainte {key}
- Préférez l'héritage aux booléens de typage en UML
- Les attributs dérivés sont réservés aux dérivations simples (des attributs de la même classe), si c'est plus complexe, préférez des méthodes (et dans le doute, précisez les modes de calcul sur le schéma ou dans une note à part)
- Donnez des exemples de contenu lorsque ce n'est pas évident (lorsque le couple nom d'attribut et type ne permet pas de façon évidente de comprendre de quoi l'on parle)

- Inutile de déclarer le type booléen en UML, utilisez-le directement comme un type de données connu

Complément

- Si tous vos héritages sont exclusifs, notez-le à part pour alléger votre schéma (et éviter l'abondance de XOR)

b) Bibliographie commentée sur la modélisation UML

Complément : Outils de modélisation UML

Il existe de nombreux outils de modélisation UML. On pourra citer :

- Dia [w_dia] : logiciel Open Source et multi-plateformes facile d'usage (qui marche néanmoins mieux sur Linux que sur Windows).
- Objecteering [w_objecteering] (version gratuite).

À voir également en Open Source : *ArgoUML*[1] ou *EclipseUML.*[2] (non testé par l'auteur).

Complément : Modélisation UML

UML2 en action [Roques04]

Pour un aperçu plus détaillé des possibilités d'expression du diagramme de classe UML, lire le chapitre 7 : Développement du modèle statique (pages 133 à 163).

On pourra notamment y trouver :

- L'association d'agrégation
- Les propriétés d'association
- L'expression de rôles dans les associations
- Les attributs de classe
- Les qualificatifs
- Les opérations (ou méthodes)

Le chapitre donne de plus des conseils méthodologiques pour la conception (voir en particulier la synthèse page 163).

On pourra également y trouver :

- Des principes de choix de modélisation entre attributs et classes et sur la segmentation des classes
- Des principes de sélection des attributs (redondance avec les associations, avec les classes, etc.)
- Des principes de sélection des associations
- Des principes de choix de cardinalité (notamment pour la gestion d'historisation)
- Des principes de sélection des relations de généralisation (héritage)
- Des principes d'introduction de métaclasses (type)s

Complément : Référence UML en ligne

UML en Français [w_uml.free.fr]

Une très bonne référence en ligne sur la modélisation UML, avec des cours, des liens vers la norme, etc.

Le contenu dépasse très largement l'usage d'UML pour la modélisation de BD (et ne fait d'ailleurs pas de référence précise à ce sous-ensemble particulier).

1 - http://argouml.tigris.org/
2 - http://www.eclipsedownload.com/

On pourra consulter en particulier le chapitre sur les diagrammes de classe : *http://uml.free.fr/cours/i-p14.html*[3]

Complément : Tutoriel sur la modélisation UML.

UML en 5 étapes [w_journaldunet.com(1)]

On consultera en particulier le tutoriel sur les diagrammes de classe : *http://developpeur.journaldunet.com/tutoriel/cpt/010607cpt_umlintro.shtml*[4]

Complément : Conseils

Cinq petits conseils pour un schéma UML efficace [w_journaldunet.com(2)]

Complément : Pratique

UML2 par la pratique [Roques09] (chapitre 3)

Des explications, exemples et études de cas.

c) Synthèse : Les diagrammes de modélisation conceptuelle

Un modèle conceptuel peut être représenté sous forme de diagramme E-A ou sous forme de diagramme de classe UML.

- Classe ou Entité
  - Attribut ou Propriété
    - Typé
    - Multi-valué
    - Composé
    - Dérivé
  - Méthode
    - Paramètres
    - Valeur de retour
- Association
  - Association
    - Verbe
    - Cardinalité
  - Héritage
    - Héritage d'attributs
    - Héritage de méthodes
  - Composition (ou entité faible)
    - Cardinalité

3 - http://uml.free.fr/cours/i-p14.html
4 - http://developpeur.journaldunet.com/tutoriel/cpt/010607cpt_umlintro.shtml

d) Correspondance entre UML et relationnel

Modèle UML	Modèle relationnel
Classe instanciable	Relation
Classe abstraite	Rien
Objet	Nuplet
Attribut simple	Attribut atomique
Attribut composite	Ensemble d'attributs
Attribut multivalué	Relation et clé étrangère
Attribut dérivé	Procédure stockée ou contrainte dynamique
Méthode	Procédure stockée
Association 1:N	Clé étrangère
Association N:M	Relation et clé étrangère
Association de degré 3 ou supérieur	Relation et clé étrangère
Classe d'association	Attributs
Composition	Clé

Tableau 3 Passsage UML vers Relationnel

4. Modélisation avancée des associations 1:1 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'association 1:1 et savoir faire le meilleur choix.

Il existe des formulations conceptuelles en UML qui sont plus délicates à traduire au niveau logique en relationnel, comme l'association 1:1. Ces cas requièrent un choix éclairé de la part du concepteur.

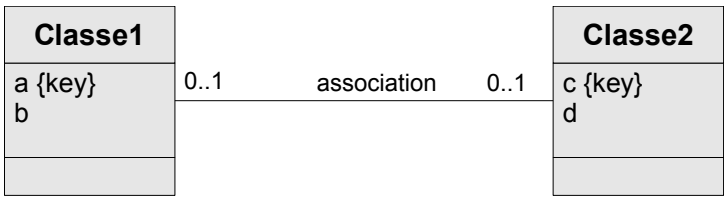
a) Association 1:1

Question

Traduire chacun des schémas conceptuels suivants en schéma relationnel.

Graphique 26 Association 1:1

Graphique 27 Association 0..1:1

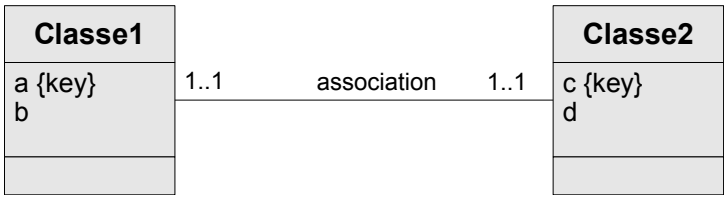


Graphique 28 Association 0..1:0..1

b) Transformation des associations 1:1 (approche générale)

Il existe deux solutions pour transformer une association 1:1 :

- **Avec deux relations** : on traite l'association 1:1 comme une association 1:N, puis l'on ajoute une contrainte UNIQUE sur la clé étrangère pour limiter la cardinalité maximale à 1 ;
- **Avec une seule relation** : on fusionne les deux classes en une seule relation.



Graphique 29 Association 1:1

Méthode : Avec deux relations (clé étrangère)

- Une des deux relations est choisie pour porter la clé étrangère ;
- on ajoute les contraintes : `UNIQUE` ou `KEY` (clé candidate) sur la clé étrangère ; et si nécessaire une contrainte imposant l'instanciation simultanée des deux relations.

```
Classe1(#a,b,c=>Classe2) avec c UNIQUE ou KEY
Classe2(#c,d)
```

Contrainte (éventuellement) : `PROJ(Classe1,c)=PROJ(Classe2,c)`

ou

```
Classe1(#a,b)
Classe2(#c,d,a=>Classe1) avec a UNIQUE ou KEY
```

Contrainte (éventuellement) : `PROJ(Classe1,a)=PROJ(Classe2,a)`

Méthode : Avec une relation (fusion)

- On créé une seule relation contenant l'ensemble des attributs des deux classes ;
- on choisit une clé parmi les clés candidates.

```
Classe12(#a,b,c,d) avec c UNIQUE ou KEY
```

ou

```
Classe21(#c,d,a,b) avec a UNIQUE ou KEY
```

Remarque : Fusion des relations dans le cas de la traduction de l'association 1:1

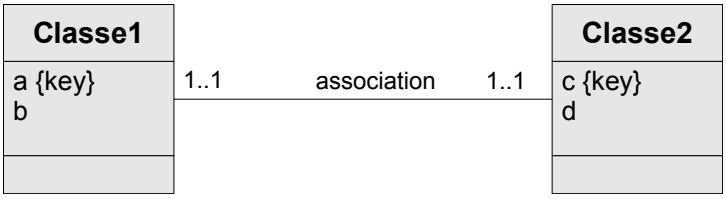
Ce choix entre les deux méthodes sera conduit par une appréciation du rapport entre :

- La complexité introduite par le fait d'avoir deux relations là ou une suffit
- La pertinence de la séparation des deux relations d'un point de vue sémantique
- Les pertes de performance dues à l'éclatement des relations
- Les pertes de performance dues au fait d'avoir une grande relation
- Les questions de sécurité et de sûreté factorisées ou non au niveau des deux relations
- ...

c) Transformation des associations 1..1:1..1

Méthode : Association 1..1:1..1

- Le plus souvent c'est méthode par **fusion** des relations qui est la plus adaptée à ce cas.
- Lorsqu'elle ne l'est pas, et que l'on choisit deux relations il faut ajouter une contrainte dynamique qui contrôlera que les deux relations sont bien toujours instanciées ensembles. Notons que la clé étrangère peut être choisie comme clé primaire.



Graphique 30 Association 1:1

`Classe12(#a,b,c,d) avec c KEY`

ou

`Classe21(#c,d,a,b) avec a KEY`

ou

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

`Classe2(#c,d)`

Contrainte : `PROJ(Classe1,c)=PROJ(Classe2,c)`

ou

`Classe1(#a,b)`

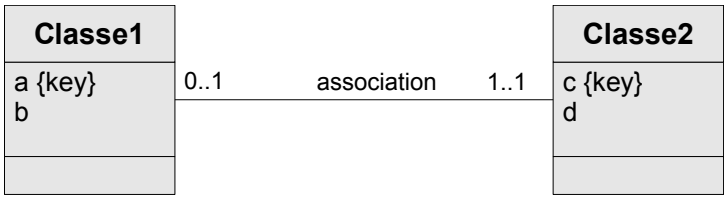
`Classe2(#c,d,a=>Classe1) avec a KEY`

Contrainte : `PROJ(Classe1,a)=PROJ(Classe2,a)`

d) Transformation des associations 0..1:1..1

Méthode : Association 0..1:1..1

- Le plus souvent c'est la méthode par les deux relations qui est la plus adaptée, **on choisira toujours la relation côté 0..1 pour être référençante**.
- Il est possible d'utiliser la fusion, dans ce cas les clés côté 0..1 deviennent des attributs `UNIQUE`, il ne peuvent plus être clés, pouvant être `NULL`.



Graphique 31 Association 0..1:1

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

ou

```
Classe12(#c,d,a,b) avec a UNIQUE
```

e) Transformation des associations 0..1:0..1

Méthode : Association 0..1:0..1

- On choisit la solution avec deux relations, d'un côté ou de l'autre ; la clé étrangère est associé à la contrainte `UNIQUE`, ce n'est pas une clé car elle peut être nulle.
- **Il n'est pas possible de choisir la fusion**, en effet l'une des deux relations pouvant être nulle, on ne pourrait plus trouver de clé.

Graphique 32 Association 0..1:0..1

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

ou

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

f) Exemple de choix pour une relation 1:1

Exemple

- Soit deux entités, "homme" et "femme" et une association "mariage" de cardinalité 1..1:1..1 entre ces deux entités (hommes et femmes sont donc obligatoirement mariés).
- Les entités "homme" et "femme" sont identifiées par un attribut "nom" (dans ce modèle chaque personne a un nom unique, on pourrait remplacer le nom par un identifiant comme le numéro de sécurité social ou par une clé artificielle pour être plus proche de la réalité).

Bien que de type 1..1:1..1, le choix de la fusion n'est pas très opportun car il s'agit bien d'objets distincts que l'on veut modéliser.

- On choisira donc plutôt la représentation avec deux relations.
- La clé étrangère pourra être du côté "homme" ou "femme", même si la pratique dominante nous incite à la mettre du côté femme.
- On pourra également décider de prendre la clé étrangère comme clé primaire.

```
1 homme (#nom)
```

```
2  femme (#mariage=>homme, nom) avec nom KEY
3  Contrainte : PROJ(homme,nom) = PROJ(femme,mariage)
```

Exemple

Si l'association avait été de cardinalité 0..1:0..1 (certains hommes et femmes ne sont pas mariés), un choix similaire se serait imposé, avec l'impossibilité de choisir la clé étrangère comme clé primaire, celle-ci pouvant être nulle et n'étant donc plus candidate.

```
1  homme (#nom)
2  femme (#nom, mariage=>homme) avec mariage UNIQUE
```

B. Exercices

1. Lab III

[20 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 dédiée de contre-indications, généralement plusieurs, parfois aucune.
- 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.

Données de test

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 :
  - Le Chourix ne doit jamais être pris après minuit.
  - Le Chourix ne doit jamais être mis au 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 :
  - Le Tropas doit être gardé à l'abri de la lumière du soleil

  Son unique composant est le **HG79**.
- Les composants existants sont :
  - **HG79** : "Vif-argent allégé"

- **HG81** : "Vif-argent alourdi"
- **SN50** : "Pur étain"

Question 1

Effectuez le modèle conceptuel en UML de ce problème.

Indices :

> *On note dans l'énoncé que les contre-indications sont **dédiée** aux médicaments, et par ailleurs on note dans les données exemples que les contre-indications sont énoncées spécifiquement par rapport aux médicaments.*
>
> *Ce n'est pas parce que les composants s'appellent ainsi dans l'énoncé que l'on est en présence d'une composition.*

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.

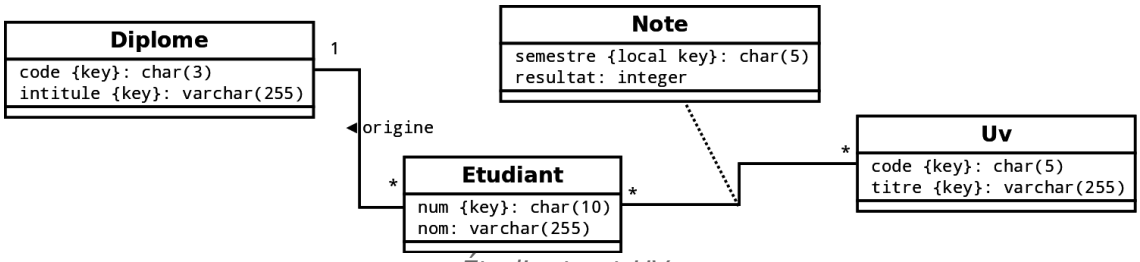
Question 3

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. Étudiants et UVs (introduction)

[20 min]

On dispose du schéma UML ci-après qui décrit des étudiants, des UV, les notes obtenues par les étudiants à ces UV, et les diplômes d'origine de ces étudiants.



Étudiants et UVs

- {key} désigne des clés candidates, ici toutes les clés ne sont composées que d'un seul attribut
- {local key} désigne une clé locale
- un semestre est de la forme 'PYYYY' ou 'AYYYY' (où YYYY désigné une année sur 4 chiffre) ; exemple : 'A2013', 'P2014...

Rappel : *Notion de clé locale dans classes d'association*

Question

Traduire le schéma en modèle logique relationnel. (`MLD1`).

On choisira obligatoirement les clés primaires parmi celles nécessitant le plus petit nombre de bits possible pour leur codage.

IV

Modélisation conceptuelle de données avancée avec le diagramme de classes UML

A. Cours

1. Diagramme de classes avancé

Objectifs

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

a) Exercice : Modéliser un site Web

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

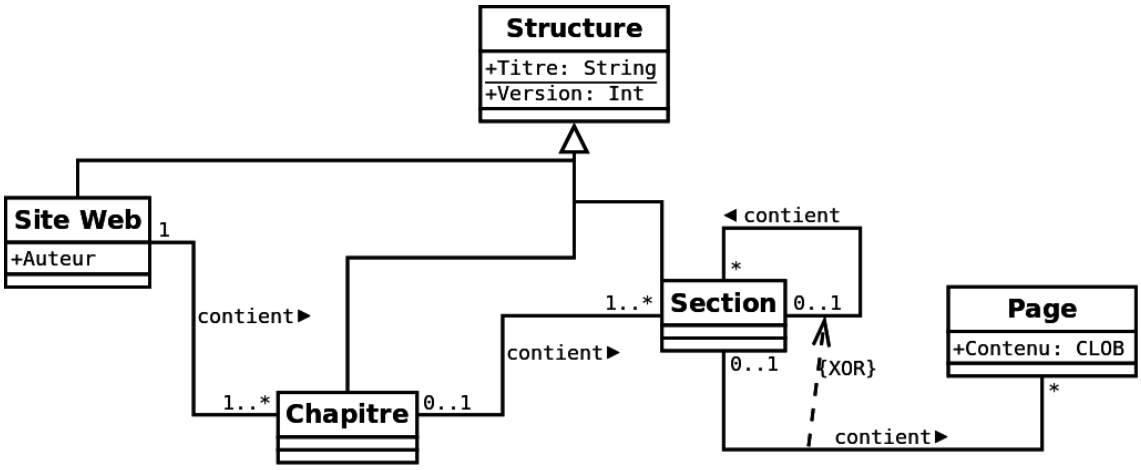


Image 18 MCD UML

☐	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.

b) 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.

Syntaxe : Notation de contraintes

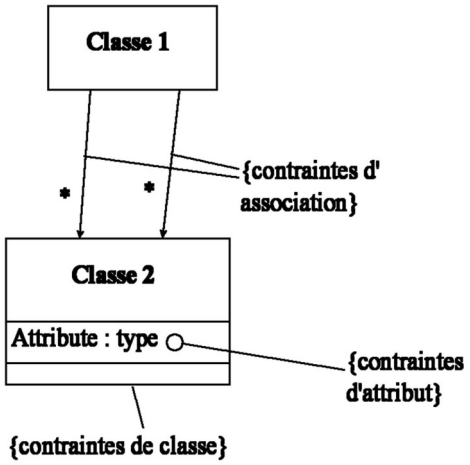


Image 19 Notation contraintes en UML

Exemple : Exemple de contraintes

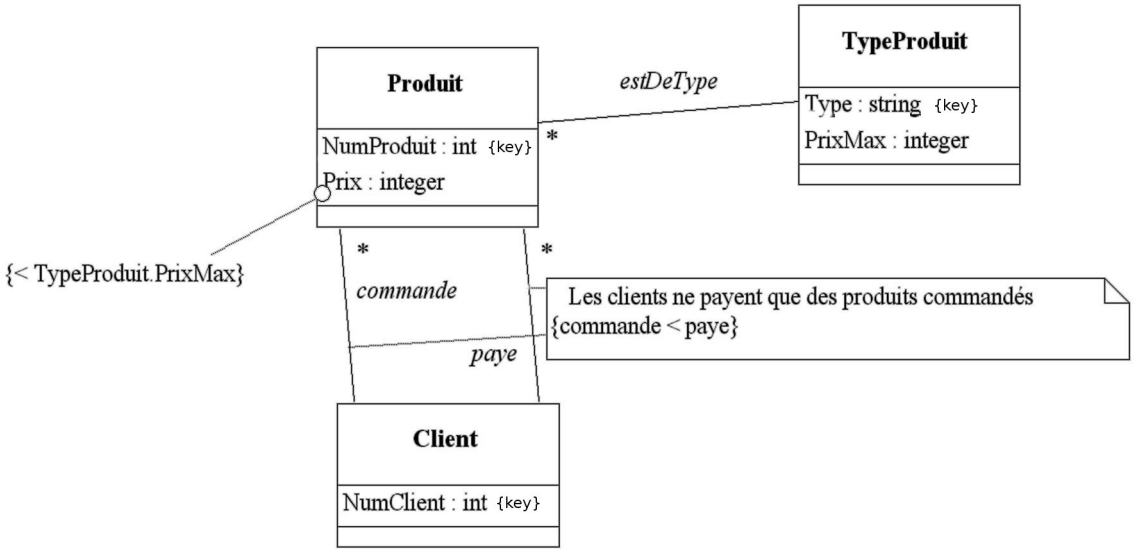


Image 20 Commandes

Méthode : Expressions formelles ou texte libre ?

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.

Méthode : Quelles contraintes exprimer ?

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.

c) Exemple de contraintes standardisées sur les associations

Rappel

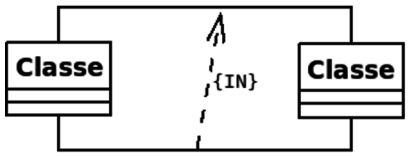


Image 21 Notation des contraintes sur les associations

Définition : Inclusion {I}

Si l'association inclue est instanciée, l'autre doit l'être aussi, la contrainte d'inclusion a un sens, représenté par une flèche (également notée {Subset} ou {IN}).

Définition : Simultanéité {S}

Si une association est instanciée, l'autre doit l'être aussi (également notée {=} ou {AND}).

La simultanéité est équivalente à une double inclusion.

Définition : Exclusion {X}

Les deux associations ne peuvent être instanciées en même temps.

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

Au moins une des deux associations doit être instanciée.

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

Exactement une des deux associations doit être instanciée.

d) Exemple de contraintes standardisées sur les attributs

Définition : {unique}

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

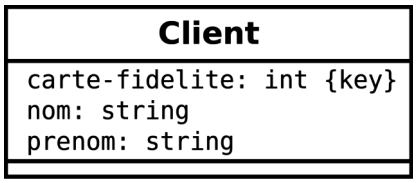
Définition : {frozen}

Une fois instancié l'attribut ne peut plus changer.

Définition : {key}

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

Exemple : Contrainte de clé



Clé en UML

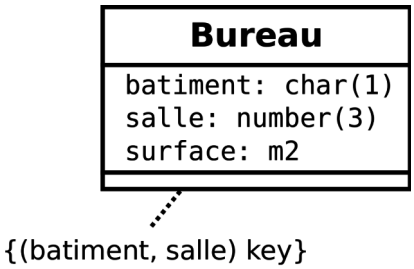


Image 22 Clé composée de deux attributs

e) Paquetages

Définition : Package

Les paquetages (plus communément appelés *package*) sont des éléments servant à organiser un modèle.

Ils sont particulièrement utiles dès que le modèle comporte de nombreuses classes et que celles-ci peuvent être triées selon plusieurs aspects structurants.

Syntaxe

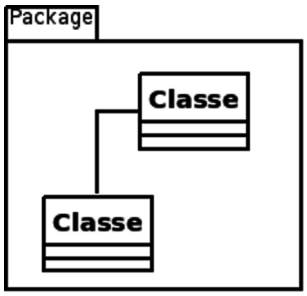


Image 23 Notation des paquetages en UML

Exemple

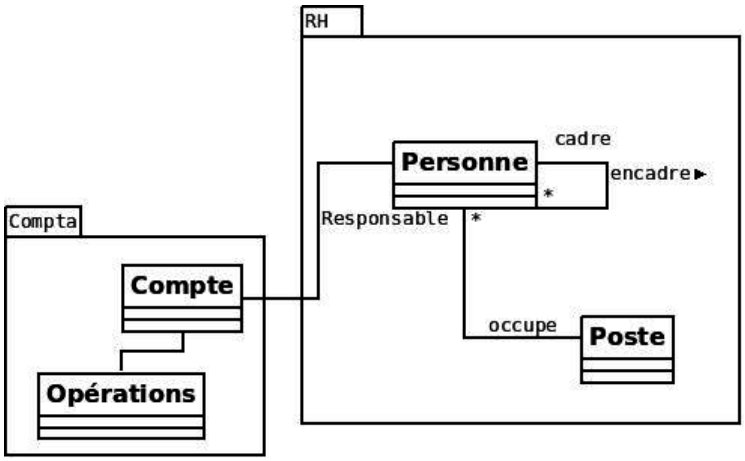


Image 24 Exemple d'utilisation des packages

Méthode

On représente chaque classe au sein d'un *package*. Il est alors possible de faire une présentation globale du modèle (tous les *packages*), partielle (une partie des *packages*) ou centrée sur un seul *package*.

Pour une représentation partielle ou centrée sur un *package*, on représente les *packages* concernés avec leurs classes propres, ainsi que toutes les classes liées des autres packages (et seulement celles-ci).

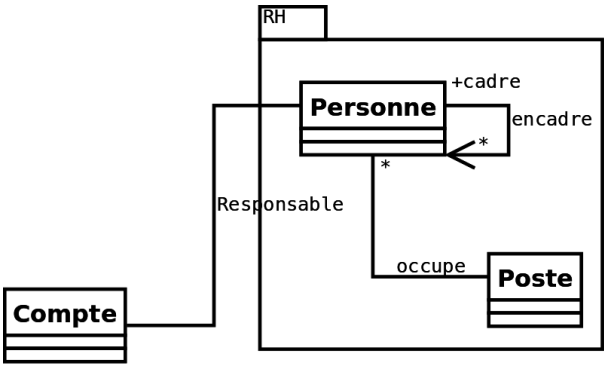


Image 25 Présentation partielle du modèle centrée sur un package

f) Stéréotype

Définition : Stéréotype UML

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.

Définition : Méta-modèle

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

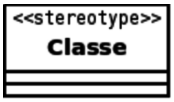


Image 26 Notation d'un stéréotype en UML

Conseil : Stéréotypes spécifiques et stéréotypes standard

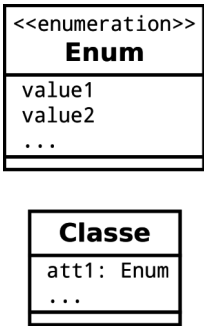
Un stéréotype spécifique enrichit le méta-modèle UML, mais selon une sémantique qui est propre à celui qui l'a posé, non standard donc. La conséquence est que pour un tiers, l'interprétation du stéréotype n'est plus normalisée, et sera potentiellement plus facilement erronée. Il convient donc de ne pas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

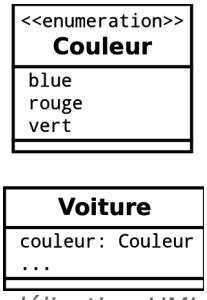
g) Énumération : stéréotype <<enumeration>>

Syntaxe



Stéréotype UML permettant d'exprimer une énumération

Exemple

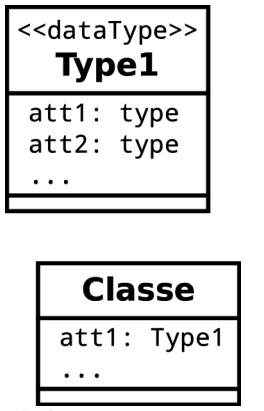


Exemple de modélisation UML d'énumération

h) Type utilisateurs : stéréotype <<dataType>>

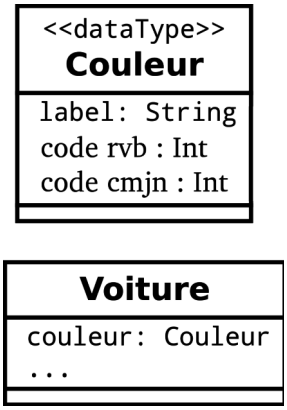
Les types utilisateurs permettent de définir des types complexes propres en extension des types primaires (entier, chaîne, date...).

Syntaxe



Stéréotype dataType

Exemple



Stéréotype dataType (exemple)

Méthode : Attributs composés

Cette modélisation est équivalente à la modélisation des attributs composés directement dans la classe principale. C'est une représentation plus standard en UML.

2. Passage UML-Relationnel : Expression de contraintes

Objectifs

Savoir exprimer les contraintes en modélisation relationnelle.

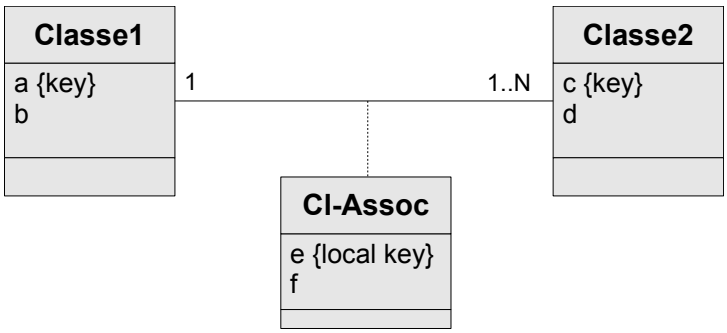
a) Contraintes

Question

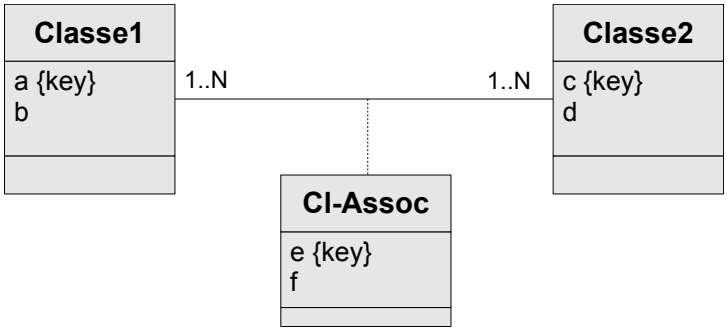
Traduire chacun des schémas conceptuels suivants en schéma relationnel.

Graphique 33 Association 1:N

Graphique 34 Association N:M



Graphique 35 Classe d'association (1:N)



Graphique 36 Classe assocation (N:M)

b) Liste des contraintes

Méthode : Contraintes exprimées sur le diagramme

Les contraintes exprimées sur le diagrammes sont reportées dans un tableau qui accompagnera le modèle logique. On ajoutera également les clés candidates.

Relation1	AND (Relation1, Relation4) ; a KEY ; ...
Relation2	b > c ; XOR (Relation2, Relation 3) ; ...

L'on peut également commenter chaque relation directement lorsque les informations ne sont pas trop nombreuses.

```
1 Relation1 (a, b, c) avec : c clé candidate ; AND(Relation1,Relation4)
2 Relation2 (b, c, d) avec : b > c ; XOR(Relation2, Relation3)
```

Méthode : Extension des contraintes exprimées

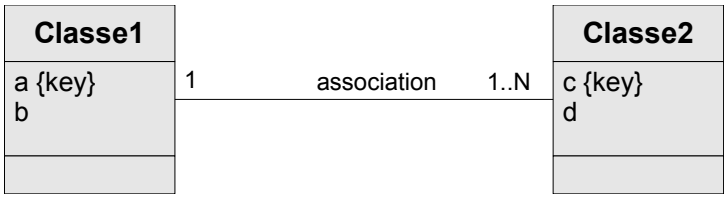
On s'attachera lors de la modélisation logique à exprimer l'ensemble des contraintes dynamiques pesant sur le modèle, même celles qui ont été considérées comme secondaires ou évidentes lors de la modélisation conceptuelle.

c) Contrainte de cardinalité minimale 1 dans les associations 1:N

Rappel

Transformation des associations 1:N

Méthode



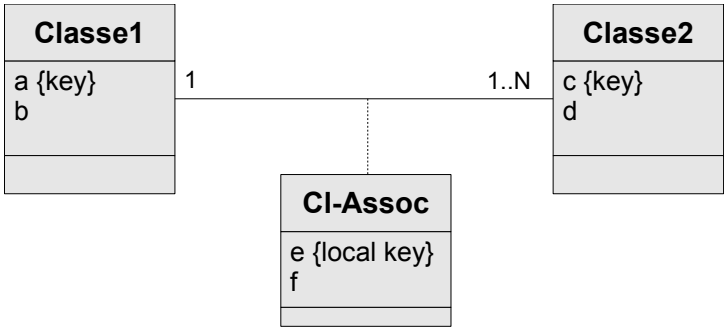
Graphique 37 Association 1:N

- 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.

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

Contraintes : `a NOT NULL et PROJECTION(Classe1,a) ⊆ PROJECTION(Classe2,a)`

Complément : Association 1:N avec classe d'association



Graphique 38 Classe d'association (1:N)

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

Contraintes : `a NOT NULL et PROJECTION(Classe1,a) ⊆ PROJECTION(Classe2,a)`

Complément

Projection

d) Contrainte de cardinalité minimale 1 dans les associations N:M

Rappel

Transformation des associations N:M

Méthode

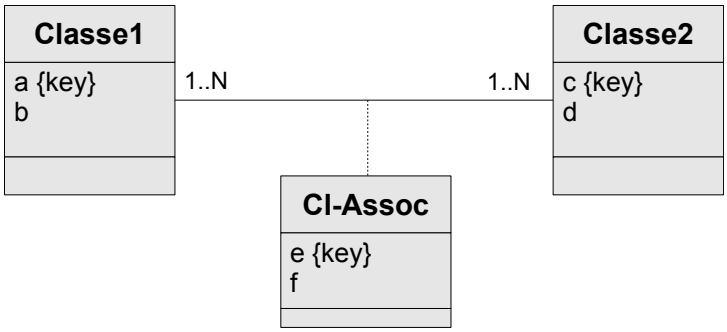
- Si la cardinalité est au moins 1 (1..N) d'un côté et/ou de l'autre, alors des contraintes d'existence simultanée de tuple devront être ajoutées.
- Ce n'est pas nécessaire si la cardinalité est 0..N.

Graphique 39 Association N:M

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

Contraintes : `PROJ(Classe1,a) ⊆ PROJ(Assoc,a) et PROJ(Classe2,c) ⊆ PROJ(Assoc,c)`

Complément : Association N:M avec classe d'association



Graphique 40 Classe assocation (N:M)

```
Classe1(#a,b)
Classe2(#c,d)
Cl-Assoc(#a=>Classe1,#c=>Classe2,#e,f)
```

Contraintes : `PROJ(Classe1,a) ⊆ PROJ(Assoc,a) et PROJ(Classe2,c) ⊆ PROJ(Assoc,c)`

Complément

Projection

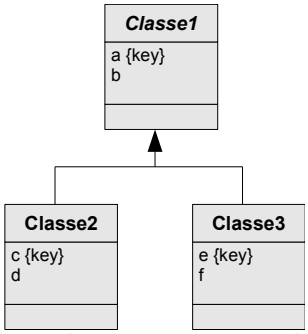
e) Contraintes de l'héritage par référence avec classe mère abstraite

Rappel

Transformation de la relation d'héritage par référence

Méthode

Si la classe mère est abstraite, il faut ajouter la contrainte que tous les tuples de la Classe1 sont référencés par un tuple de la Classe2 et/ou de la Classe3.



Graphique 41 Héritage (classe mère abstraite)

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

Contraintes : `PROJ(Classe1,a) IN (PROJ(Classe2,a) UNION PROJ(Classe3,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
```

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.

f) Contraintes de l'héritage par les classes filles avec classe mère non abstraite

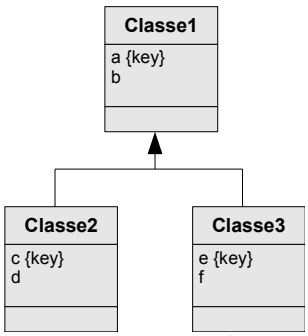
Rappel

Transformation de la relation d'héritage par les classes filles

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 exister dans les classes filles



Graphique 42 Héritage

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

Contrainte : `PROJ(Classe1,a) NOT IN (PROJ(Classe2,a) UNION PROJ(Classe3,a))`

Exemple : Héritage absorbé par les classes filles

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)
```

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.

g) Contraintes de l'héritage par la classe mère avec classe mère abstraite

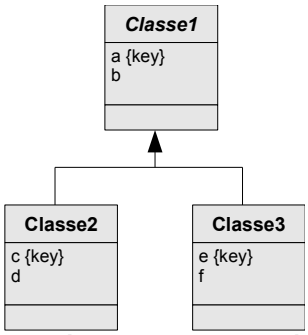
Rappel

Transformation de la relation d'héritage par la classe mère

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 (ou les deux).



Graphique 43 Héritage (classe mère abstraite)

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

Contraintes :

- `c UNIQUE et e UNIQUE`
- `AND (c NOT NULL OR e NOT NULL)`
- `AND t NOT NULL`

Exemple : Héritage absorbé par la classe mère

Soit la classe 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  A (#K, A1, A2, B1, B2, C1, C2, T:{'B','C'})
```

Si l'on pose que A n'est pas abstraite, alors un tuple sera un A s'il a la valeur null pour sa propriété D. Si l'on pose que A est abstraite, on ajoutera la contrainte NOT NULL à la propriété T. On peut aussi ajouter 'A' aux valeurs possible de T.

Complément : Héritage exclusif

Si l'héritage est exclusif, que la classe mère soit abstraite ou non, il faudrait vérifier par des contraintes que l'attribut de discrimination T et les attributs valués sont en correspondance, afin d'empêcher toute incohérence :

- `(T=2 AND c)`
- `(T=3 AND e)`
- `NOT (c AND e)`

- `NOT (c AND f)`
- `...`

B. Exercices

1. Ouaf !

[45 min]

Un réseau de chenils souhaite informatiser son activité et vous demande de créer un modèle conceptuel de sa future base de données. Un chenil est un établissement destiné à l'élevage ou à la pension des chiens. Dans ce réseau, les chenils s'occupent des chiens de leur naissance jusqu'à la fin de leur apprentissage. L'identité du propriétaire étant privée, aucun chien ne sera relié à un propriétaire dans cette base de données qui sert aussi de vitrine sur le savoir-faire du réseau.

Un chenil du réseau possède un nom unique permettant de l'identifier, un nom de contact ainsi qu'un numéro de téléphone, et une description de ses activités. Chaque chien est identifié par son nom et celui du chenil qui gère son apprentissage ; il possède une date de naissance et est associé à une race. Certains des chiens suivront un apprentissage plus spécifique : ils deviendront guides d'aveugle, chiens de garde ou chiens de course. Chaque chien de garde possède une spécialité : attaque, défense, pistage ou détection. Un chien de course quand à lui possède une vitesse maximum mesurée.

Le chenil gère également l'historique des poids de chaque animal, afin d'enregistrer régulièrement la courbe de croissance. Pour cela chaque relevé de poids est identifié par le nom du chien et la date du relevé.

Les chiens aux parcours plus spécifiques (guides, chiens de garde ou chiens de course) sont liés à des entités qui les géreront après l'apprentissage. Ces entités sont identifiées par un nom, et possèdent une description ainsi qu'un nom de contact avec un numéro de téléphone. Les chiens guides d'aveugle sont ainsi gérés par une association qui propose ensuite les chiens aux personnes malvoyantes. Les chiens de gardes sont utilisés par des entreprises (le chenil souhaite pouvoir ajouter une description de l'activité principale de chaque entreprise). Et un chien de course est possédé par une écurie (qui a une date de création). Bien sûr, chacune de ces entités peut gérer plusieurs animaux.

Question 1

Réalisez un modèle conceptuel en UML répondant aux besoins du réseau de chenils, sous la forme d'un package **Chenil**.

Le réseau souhaiterait compléter cette première modélisation en ajoutant des informations sur les chiens de course, pour améliorer sa publicité. L'objectif est de garder en mémoire pour chaque épreuve sportive auquel participe un chien de course son numéro de dossard (de 1 à 6), sa position à l'issue de l'épreuve (de 1 à 6), son temps de course, et s'il a abandonné ou non.

Une épreuve est identifiée par la date et le nom du tournoi à laquelle elle appartient (un tournoi est identifié uniquement par son nom). Il y a toujours 6 chiens qui courent dans une épreuve, mais ils ne proviennent pas tous d'un des chenils du réseau (nous ne gérons pas les chiens en dehors du réseau). Chaque épreuve a lieu dans un cynodrome identifié par son nom, et chaque cynodrome se situe dans un département, identifié par son numéro et possédant un nom.

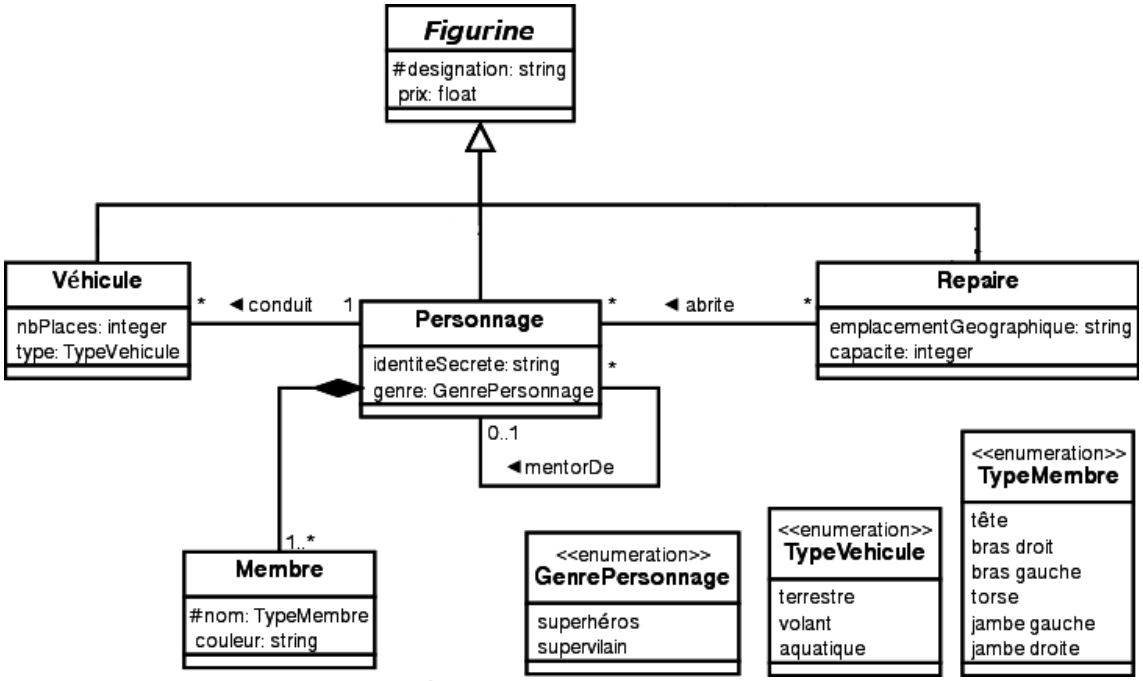
Question 2

Réalisez un second diagramme UML répondant à ces nouveaux besoins, sous la forme d'un second package **Course**.

2. Super-héros relationnels I

[20 min]

La gamme de super-héros GARVEL veut réaliser la base de données de leurs figurines articulées. La société a fait réaliser un modèle UML qui doit servir de point de départ à la mise en œuvre.



Modèle UML Figurines GARVEL

Question

Transformer le modèle UML en modèle relationnel (justifier les passages non triviaux, en particulier la relation d'héritage).

3. Médiathèque

[45 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.

Analyse de bases de données SQL avec les agrégats (GROUP BY)

V

A. Cours

1. Introduction aux agrégats avec GROUP BY

Objectifs

Savoir réaliser un agrégat en SQL en utilisant les fonctions de regroupement et les clause GROUP BY

a) Exercice : La somme finale

Que renvoie la dernière instruction SQL de la séquence ci-dessous ?

```
1  CREATE TABLE R1 (a INTEGER, b INTEGER);
2  CREATE TABLE R2 (b INTEGER);
3  INSERT INTO R1 VALUES (1,1);
4  INSERT INTO R1 VALUES (2,1);
5  INSERT INTO R2 VALUES (1);
6  INSERT INTO R2 VALUES (2);
7  SELECT sum(R1.a) FROM R1, R2 WHERE R1.b=R2.b;
```

b) Définition de l'agrégation

Définition : Agrégat

Un agrégat est un partitionnement horizontal d'une table en sous-tables, en fonction des valeurs d'un ou plusieurs attributs de partitionnement, suivi éventuellement de l'application d'une fonction de calcul à chaque attribut des sous-tables obtenues.

Synonyme : Regroupement

Syntaxe

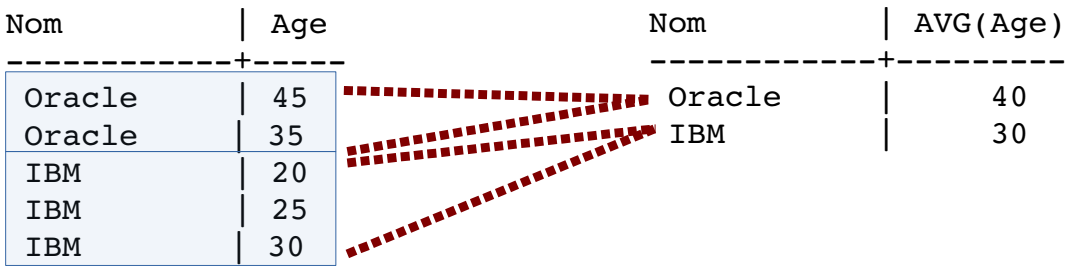
```
1  SELECT liste d'attributs de partionnement à projeter et de fonctions de calcul
2  FROM liste de relations
3  WHERE condition à appliquer avant calcul de l'agrégat
4  GROUP BY liste ordonnée d'attributs de partitionnement
```

1. La table est divisée en sous-ensembles de lignes, avec un sous-ensemble pour chaque valeur différente des attributs de partitionnement projetés dans le SELECT
2. Les fonctions d'agrégation sont appliquées sur les attributs concernés

Exemple

```
1  SELECT Societe.Nom, AVG(Personne.Age)
2  FROM Personne, Societe
3  WHERE Personne.NomSoc = Societe.Nom
4  GROUP BY Societe.Nom
```

Societe.Nom est un ici le seul attribut de partitionnement, donc un sous ensemble est créé pour chaque valeur différente de *Societe.Nom*, puis la moyenne (fonction AVG) est effectuée pour chaque sous-ensemble.



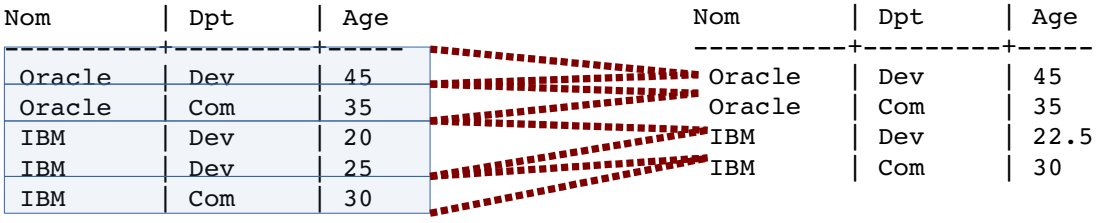
Regroupement avec un attribut et une fonction

Cette requête calcul l'âge moyen du personnel pour chaque société.

Exemple

```
1  SELECT Societe.Nom, Societe.Dpt AVG(Personne.Age)
2  FROM Personne, Societe
3  WHERE Personne.NomSoc = Societe.Nom
4  GROUP BY Societe.Nom, Societe.Dpt
```

Societe.Nom et *Societe.Dpt* sont les deux attributs de partitionnement, donc un sous-ensemble est créé pour chaque valeur différente du couple (*Societe.Nom*,*Societe.Dpt*).



Regroupement avec deux attributs et une fonction

Cette requête calcul l'âge moyen du personnel pour chaque département de chaque société.

Remarque : Requête inutile

La requête est inutile dès lors que l'agrégat est défini sur une valeur unique dans la relation, puisqu'on aura bien une ligne par ligne de la table source.

```
1  SELECT Societe.Nom
```

```
2  FROM Societe
3  GROUP BY Societe.Nom
```

```
countrycode          countrycode
-------------        -------------
 Oracle               Oracle
 IBM                  IBM
```

Exemple d'agrégat inutile (countrycode est une clé de country)

c) Exemple d'attribut d'agrégation

Schéma relationnel

```
1  country(#countrycode:char(2), name:varchar, population:numeric)
2  city(#code:char(3), countrycode=>country, name:varchar, population:numeric):
```

Schéma de base de données

```
1  CREATE TABLE country (
2  countrycode CHAR(2) NOT NULL,
3  name VARCHAR NOT NULL,
4  population NUMERIC(3),
5  PRIMARY KEY (countrycode)
6  );
7
8  CREATE TABLE city (
9  citycode CHAR(3) NOT NULL,
10 countrycode CHAR(2) NOT NULL,
11 name VARCHAR NOT NULL,
12 population NUMERIC(2,1),
13 PRIMARY KEY (citycode),
14 FOREIGN KEY (countrycode) REFERENCES country(countrycode)
15 );
```

Données

```
1  INSERT INTO country VALUES ('ES', 'Spain', 46);
2  INSERT INTO country VALUES ('FR', 'France', 67);
3  INSERT INTO country VALUES ('DE', 'Germany', 82);
4
5  INSERT INTO city VALUES ('BAR', 'ES', 'Barcelona', 1.9);
6  INSERT INTO city VALUES ('MAD', 'ES', 'Madrid', 3.3);
7  INSERT INTO city VALUES ('ZAR', 'ES', 'Zaragoza', 0.7);
8
9  INSERT INTO city VALUES ('PAR', 'FR', 'Paris', 2.2);
10 INSERT INTO city VALUES ('LYO', 'FR', 'Paris', 0.5);
11 INSERT INTO city VALUES ('LLL', 'FR', 'Lille', 0.2);
12 INSERT INTO city VALUES ('AMN', 'FR', 'Amiens', 0.1);
```

Sélectionne les countrycode existants dans la table city

```
1  SELECT countrycode
2  FROM city;
```

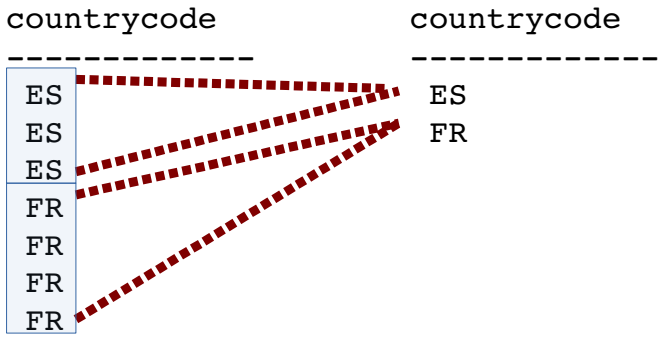
```
1   countrycode
2  -------------
3   ES
4   ES
5   ES
```

```
6   FR
7   FR
8   FR
9   FR
```

Sélectionne les countrycode existants dans la table city avec agrégat

```
1  SELECT countrycode
2  FROM city
3  GROUP BY countrycode;
```

```
1   countrycode
2  -------------
3   FR
4   ES
```



Principe de l'agrégation

d) Fonctions d'agrégation

Définition : Fonctions d'agrégation

Une fonction d'agrégation (ou fonction de regroupement) s'applique aux valeurs du sous-ensemble d'un agrégat e relation avec pour résultat la production d'une valeur atomique unique (entier, chaîne, date, etc).

Les cinq fonctions prédéfinies sont :

- **Count(Relation.Propriété)**

  Renvoie le nombre de valeurs non nulles d'une propriété pour tous les tuples d'une relation ;
- **Sum(Relation.Propriété)**

  Renvoie la somme des valeurs d'une propriété des tuples (numériques) d'une relation ;
- **Avg(Relation.Propriété)**

  Renvoie la moyenne des valeurs d'une propriété des tuples (numériques) d'une relation ;
- **Min(Relation.Propriété)**

  Renvoie la plus petite valeur d'une propriété parmi les tuples d'une relation .
- **Max(Relation.Propriété)**

  Renvoie la plus grande valeur d'une propriété parmi les tuples d'une relation.

Attention : Fonctions de calcul sans partitionnement

Si une ou plusieurs fonctions de calcul sont appliquées sans partitionnement, le résultat de la requête est un tuple unique.

Exemple

```
1  SELECT Min(Age), Max(Age), Avg(Age)
2  FROM Personne
3  WHERE Qualification='Ingénieur'
```

Remarque : Comptage d'une relation

Pour effectuer un comptage sur tous les tuples d'une relation, appliquer la fonction `count` à un attribut de la clé primaire. En effet cet attribut étant non nul par définition, il assure que tous les tuples seront comptés.

e) Exemple de fonctions d'agrégation

Rappel : Schéma relationnel

```
1  country(#countrycode:char(2), name:varchar, population:numeric)
2  city(#code:char(3), countrycode=>country, name:varchar, population:numeric):
```

Exemple d'attribut d'agrégation

i Agrégat avec un attribut d'agrégation et une fonction d'agrégation

Requête sans agrégat

Sélectionne les *countrycode* et les *citycode* existants dans la table *city*.

```
1  SELECT countrycode, citycode
2  FROM city;
```

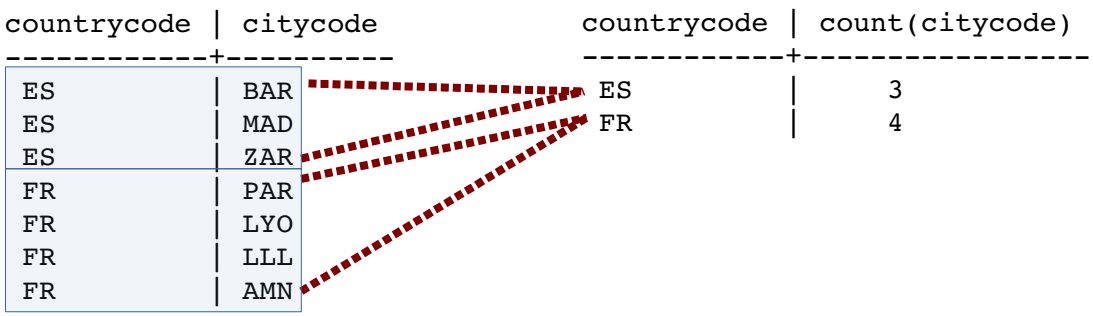
```
1   countrycode | citycode
2  -------------+----------
3   ES          | BAR
4   ES          | MAD
5   ES          | ZAR
6   FR          | PAR
7   FR          | LYO
8   FR          | LLL
9   FR          | AMN
```

Requête avec agrégat

- Sélectionne les *countrycode* et les *citycode* existants dans la table *city*,
- puis agrège par valeurs distinctes de *countrycode*.

```
1  SELECT countrycode, count(citycode)
2  FROM city
3  GROUP BY countrycode;
```

```
1   countrycode | count
2  -------------+-------
3   FR          |     4
4   ES          |     3
```



Regroupement avec fonction d'agrégation

ii Agrégat avec un attribut d'agrégation et deux fonctions d'agrégation

Sélectionne les *countrycode*, les *citycode* et les *populations* existants dans la table *city*.

```
1  SELECT countrycode, citycode, population
2  FROM city;
```

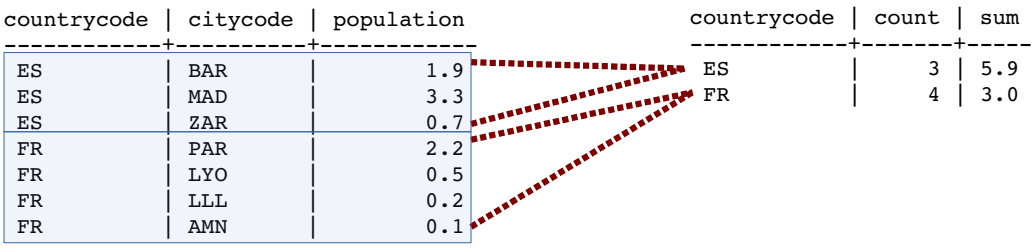
```
1  countrycode | citycode | population
2  ------------+----------+------------
3  ES          | BAR      |        1.9
4  ES          | MAD      |        3.3
5  ES          | ZAR      |        0.7
6  FR          | PAR      |        2.2
7  FR          | LYO      |        0.5
8  FR          | LLL      |        0.2
9  FR          | AMN      |        0.1
```

Requête avec agrégat

- Sélectionne les *countrycode*, les *citycode* et les *populations* existants dans la table *city*,
- puis agrège par valeurs distinctes de *countrycode*
- puis calcule les fonctions *count* et *sum*.

```
1  SELECT countrycode, count(citycode), sum(population)
2  FROM city
3  GROUP BY countrycode;
```

```
1  countrycode | count | sum
2  ------------+-------+-----
3  FR          |     4 | 3.0
4  ES          |     3 | 5.9
```



Regroupement avec deux fonctions d'agrégation

iii Agrégat sans attribut d'agrégation et avec une fonction d'agrégation

Requête sans agrégat

Sélectionne les *populations* existants dans la table *city*.

```
1  SELECT population
2  FROM city;
```

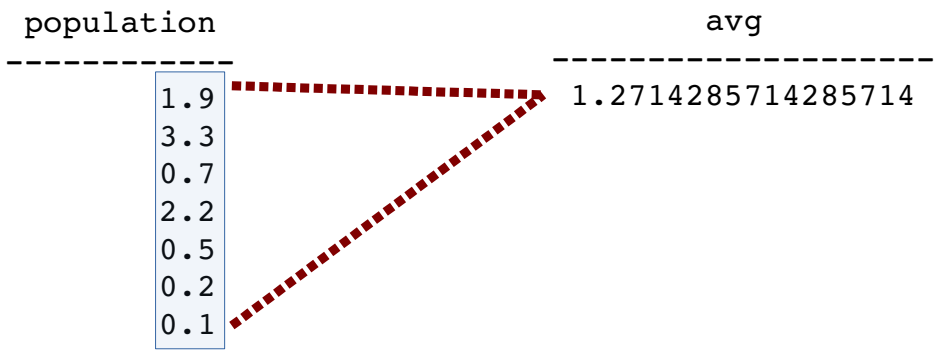
```
1   population
2  ------------
3          1.9
4          3.3
5          0.7
6          2.2
7          0.5
8          0.2
9          0.1
```

Requête avec agrégat

- Sélectionne les *populations* existants dans la table *city*,
- puis calcule la fonction *avg* (pour *average*, moyenne).

```
1  SELECT avg(population)
2  FROM city;
```

```
1          avg
2  --------------------
3   1.2714285714285714
```



Regroupement avec fonction d'agrégation sans GROUP BY

2. Approfondissement des agrégats avec GROUP BY et HAVING

Objectifs

Savoir réaliser un agrégat en SQL en utilisant les fonctions de regroupement et les clause GROUP BY et HAVING

a) Single-Value Rule

Fondamental : Principe de la "Single-Value Rule" établie par le standard SQL

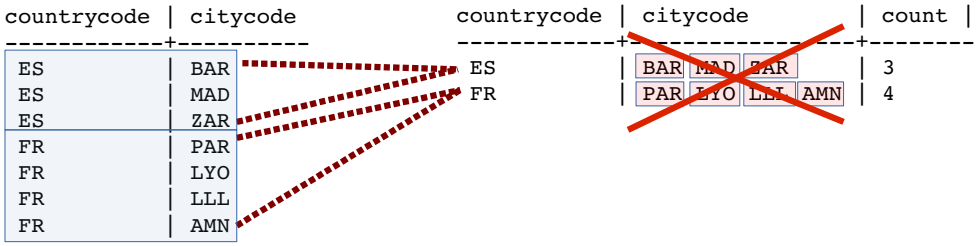
Toute colonne de la clause GROUP BY doit désigner une colonne présente dans la clause SELECT :

- **soit comme attribut d'agrégation,**
- **soit comme attribut présent dans une fonction d'agrégation.**

Attention : Requête illégale

```
1  SELECT countrycode, citycode, COUNT(citycode)
2  FROM city
3  GROUP BY countrycode;
```

```
1  ERROR:  column "city.citycode" must appear in the GROUP BY clause or be used in
       an aggregate function;
```



Tentative de GROUP BY illégal

La requête est non standard et non logique car on cherche à mettre plusieurs données dans la case *citycode* après le GROUP BY (il y a plusieurs *citycode* par *countrycode*).

Elle sera refusée par tous les SGBD.

Complément : Single-Value Rule

https://mariadb.com/kb/en/sql-99/the-single-value-rule[5]

i Tolérance (de certains SGBD)

Requête non standard tolérée par certains SGBD

```
1  SELECT citycode, countrycode, AVG(population)
2  FROM city
3  GROUP BY citycode;
```

```
1   citycode | countrycode |          avg
2  ----------+-------------+-----------------------
```

5 - https://mariadb.com/kb/en/sql-99/the-single-value-rule/

```
3  LLL      | FR          | 0.20000000000000000000
4  BAR      | ES          | 1.90000000000000000000
5  PAR      | FR          |     2.2000000000000000
6  LYO      | FR          | 0.50000000000000000000
7  ZAR      | ES          | 0.70000000000000000000
8  AMN      | FR          | 0.10000000000000000000
9  MAD      | ES          |     3.3000000000000000
```

La requête est non standard, même si c'est logiquement acceptable, car il n'y a qu'un *countrycode* par *citycode* ici, *city* étant une clé.

Certains SGBD comme Postgres accepte donc cette syntaxe, en ajoutant implicitement l'attribut qui manque au GROUP BY, car il n'y a pas d'ambiguïté si l'on analyse le graphe des DF✯ : *citycode* → *countrycode*. Mais le standard impose d'être explicite, car le SGBD a le droit de ne pas le deviner.

ii Tolérance partielle

Requête légale

```
1  SELECT ci.countrycode, co.name, count(*)
2  FROM city ci JOIN country co
3  ON ci.countrycode=co.countrycode
4  GROUP BY ci.countrycode, co.name;
```

```
1   countrycode |  name   | count
2  -------------+---------+-------
3   FR          | France  |     4
4   ES          | Spain   |     3
```

Requête non standard tolérée par certains SGBD

```
1  SELECT co.countrycode, co.name, count(*)
2  FROM city ci JOIN country co
3  ON ci.countrycode=co.countrycode
4  GROUP BY co.countrycode;
```

La requête est non standard, car *co.name* est dans le SELECT mais pas dans le GROUP BY. Comme dans l'exemple précédent certains SGBD accepteront cette requête grâce à la DF évidente : *co.countrycode* → *co.name*

Attention : Requête non standard non tolérée

Mais si l'on utilise à présent *ci.countrycode* à la place de *co.countrycode*...

```
1  SELECT ci.countrycode, co.name, count(*)
2  FROM city ci JOIN country co
3  ON ci.countrycode=co.countrycode
4  GROUP BY ci.countrycode;
```

```
1  ERROR:  column "co.name" must appear in the GROUP BY clause or be used in an
   aggregate function
```

La requête est non standard, *co.name* doit être mentionné dans la clause GROUP BY. Logiquement on pourrait juger cela superflu, car par transitivité *ci.countrycode* → *co.name*, mais le SGBD (ici Postgres) ne va pas jusque là.

Conseil

Il est donc conseillé d'appliquer le *Single-Value Rule* et de toujours expliciter tous les attributs dans le GROUP BY, pour éviter la dépendance au SGBD ou bien les erreurs liées à des variations mineures de syntaxe.

b) Restriction après agrégation (HAVING)

Définition

La clause HAVING permet d'effectuer une second restriction après l'opération d'agrégation.

Syntaxe

```
SELECT liste d'attributs de partionnement à projeter et de fonctions de calcul
FROM liste de relations
WHERE condition à appliquer avant calcul de l'agrégat
GROUP BY liste ordonnée d'attributs de partitionnement
HAVING condition sur les fonctions de calcul
```

Exemple

```
SELECT Societe.Nom, AVG(Personne.Age)
FROM Personne, Societe
WHERE Personne.NomSoc = Societe.Nom
GROUP BY Societe.Nom
HAVING COUNT(Personne.NumSS) > 2
```

Cette requête calcul l'âge moyen du personnel pour chaque société comportant plus de 2 salariés.

c) Ordre de résolution des requête SQL

Fondamental

L'ordre de résolution standard d'une requête SQL est :

1. **FROM**
2. **WHERE**
3. **GROUP BY**
4. **HAVING**
5. **SELECT**
6. **ORDER BY**

Attention : Usage des alias dans le GROUP BY ou le HAVING : requête non standard, acceptée par certains SGBD

```
SELECT substring(name,1,1) AS initiale, count(citycode)
FROM city
GROUP BY initiale;
```

```
 initiale | count
----------+-------
 L        |     1
 Z        |     1
 B        |     1
 M        |     1
 P        |     2
 A        |     1

```

La requête n'est pas standard, les résolutions de l'alias et de la fonction *substring* sont dans le SELECT, qui est postérieur à la résolution du GROUP BY. Postgres acceptera néanmoins cette syntaxe, ce qui évite de créer une vue ou d'utiliser une sous-requête.

Remarque : Restriction

Une restriction peut être appliquée avant calcul de l'agrégat, au niveau de la clause WHERE, portant ainsi sur la relation de départ, mais aussi après calcul de l'agrégat sur les résultats de ce dernier, au niveau de la clause HAVING.

Remarque : Utilisation de fonctions d'agrégation

Les fonctions d'agrégation peuvent être utilisées dans la claude HAVING ou dans la clause ORDER BY. Les fonctions **ne peuvent pas** être utilisées dans la clause WHERE (qui est résolu avant le GROUP BY).

d) Re-représentation de représentants

[30 minutes]

Soit le schéma relationnel suivant :

```
REPRESENTANTS (#NR, NOMR, VILLE)
PRODUITS (#NP, NOMP, COUL, PDS)
CLIENTS (#NC, NOMC, VILLE)
VENTES (#NR=>REPRESENTANTS(NR), #NP=>PRODUITS(NP), #NC=>CLIENTS(NC), QT)
```

Écrire en SQL les requêtes permettant d'obtenir les informations suivantes.

```
/* Les requêtes peuvent être testées dans un SGBDR, en créant une base de données
 avec le script SQL suivant */

/*
DROP TABLE VENTES ;
DROP TABLE CLIENTS ;
DROP TABLE PRODUITS ;
DROP TABLE REPRESENTANTS ;
*/

CREATE TABLE REPRESENTANTS (
 NR INTEGER PRIMARY KEY,
 NOMR VARCHAR,
 VILLE VARCHAR
);

CREATE TABLE PRODUITS (
 NP INTEGER PRIMARY KEY,
 NOMP VARCHAR,
 COUL VARCHAR,
 PDS INTEGER
);

CREATE TABLE CLIENTS (
 NC INTEGER PRIMARY KEY,
 NOMC VARCHAR,
 VILLE VARCHAR
);

CREATE TABLE VENTES (
 NR INTEGER REFERENCES REPRESENTANTS(NR),
 NP INTEGER REFERENCES PRODUITS(NP),
 NC INTEGER REFERENCES CLIENTS(NC),
 QT INTEGER,
 PRIMARY KEY (NR, NP, NC)
);

INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (1, 'Stephane', 'Lyon');
INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (2, 'Benjamin', 'Paris');
INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (3, 'Leonard', 'Lyon');
INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (4, 'Antoine', 'Brest');
INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (5, 'Bruno', 'Bayonne');
```

```
42
43 INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (1, 'Aspirateur', 'Rouge',
   3546);
44 INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (2, 'Trottinette', 'Bleu',
   1423);
45 INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (3, 'Chaise', 'Blanc', 3827);
46 INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (4, 'Tapis', 'Rouge', 1423);
47
48 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (1, 'Alice', 'Lyon');
49 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (2, 'Bruno', 'Lyon');
50 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (3, 'Charles', 'Compiègne');
51 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (4, 'Denis', 'Montpellier');
52 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (5, 'Emile', 'Strasbourg');
53
54 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (1, 1, 1, 1);
55 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (1, 1, 2, 1);
56 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (2, 2, 3, 1);
57 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (4, 3, 3, 200);
58 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 4, 2, 190);
59 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (1, 3, 2, 300);
60 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 2, 120);
61 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 4, 120);
62 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 4, 4, 2);
63 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 1, 3);
64 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 4, 1, 5);
65 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 3, 1);
```

Question 1

Le nombre de clients.

Question 2

Le nombre de produits de couleur rouge.

Question 3

Le nombre de clients par ville.

Question 4

La quantité totale des produits rouge vendus par chaque représentant.

Question 5

La quantité totale de produits rouges vendus pour chaque représentant ayant vendu plus de 5 fois des produits rouges (ayant réalisé plus de 5 ventes différentes de produits rouges).

B. Exercices

1. Location d'appartements en groupe

[20 min]

Soit le schéma relationnel suivant gérant le fonctionnement d'une agence de location d'appartements.

```
1 APPARTEMENT(#code_appt:String, adresse:String, type:{studio,F1,F2,F3,F4,F5+},
  prix_loyer:Real)
2 LOCATAIRE(#code_loc:String, nom:String, prenom:String)
3 LOCATION(#code_loc=>Locataire, #code_appt=>Appartement)
4 PAIEMENT_LOYER(#code_loc=>Locataire, #code_appt=>Appartement,
  #date_payement:Date, prix_paye:Real)
```

Question 1

En SQL afficher le nombre d'appartements de chaque type, uniquement pour les types qui commencent par la lettre F.

Question 2

En SQL afficher le total payé par locataire (avec son code, nom et prenom) pour l'ensemble de ses appartements.

Question 3

En SQL afficher les locataires (code uniquement) qui louent au moins 2 appartements, en précisant le nombre d'appartements loués et la moyenne des loyers, et trié par ordre décroissant de cette moyenne.

2. Championnat de Formule 1

[20 min]

La base de données suivante permet de gérer les résultats des courses de Formule 1 dans un championnat.

```
1  CHAMPIONNAT(#nom:string, annee:integer)
2  CIRCUIT(#nom:string, ville:string)
3  COURSE(#nom:string,circuit=>CIRCUIT(nom), championnat=>CHAMPIONNAT(nom))
4  SPONSOR(#nom:string)
5  ECURIE(#nom:string, devise:string, couleur:string, sponsor=>SPONSOR(nom))
6  PILOTE(#id:integer, nom:string, prenom:string, ecurie=>ECURIE(nom))
7  TOUR(#pilote => PILOTE(id), #course => COURSE(nom), #num:integer, duree:integer)
```

Question 1

En algèbre relationnel et en SQL, afficher la liste de tous les pilotes dont le nom commence par la lettre 'D'.

Question 2

En SQL, afficher le nombre de pilotes par écurie en classant les résultats par ordre alphabétique des noms des écuries.

Question 3

En algèbre relationnel et en SQL, afficher les noms des sponsors qui ne sont pas liés à une écurie.

Question 4

En SQL, afficher le numéro, nom, prénom et écurie, avec leur temps moyen par tour, des pilotes participant à la course `Daytonutc 500` ; mais en ne conservant que les pilotes qui ont effectués au moins deux tours de piste, et en classant le résultat par temps moyen décroissant.

3. Questions scolaires

[30 min]

Soit le schéma relationnel suivant gérant le fonctionnement d'une école et les notes des élèves.

```
1  CLASSE(#intitule)
2  MATIERE(#intitule)
3  ELEVE(#login, nom, prenom, age, classe=>CLASSE)
4  ENSEIGNANT(#login, nom, prenom, mail, matiere=>MATIERE)
5  ENSEIGNE(#enseignant=> ENSEIGNANT, #classe=>CLASSE)
```

```
6  NOTE(#eleve=>ELEVE, #enseignant=>ENSEIGNANT, #date_examen, note)
7
8  Contrainte : un enseignant ne peut mettre une note à un élève que si celui-ci se
   trouve dans une classe dans laquelle il enseigne.
```

NB : Nous n'utiliserons pas de sous-requêtes.

Question 1

En algèbre relationnel **et** en SQL, afficher la liste des tous les étudiants dont le nom commence par A.

Question 2

En algèbre relationnel **et** en SQL, afficher les noms et prénoms des enseignants qui n'enseignent à aucune classe.

Question 3

En SQL, affichez le nombre d'étudiants enregistrés en "Terminale S 2".

Question 4

En SQL, affichez les logins, noms, prénoms, classes et moyennes des élèves en cours de "Mathématiques", par ordre décroissant de moyenne, à condition qu'ils aient au minimum 2 notes dans cette matière.

Question 5

En SQL, à des fins de statistiques, nous souhaitons rechercher par enseignant et par classe, les classes qui n'ont pas la moyenne générale, et afficher pour celles-ci : le nom, prénom et mail de l'enseignant en question, la matière enseignée, la classe, la moyenne d'âge des étudiants avec les extrêmes (minimum et maximum), la moyenne générale de la classe avec les valeurs extrêmes, ainsi que le nombre d'étudiants présents dans cette classe ; le tout classé par ordre alphabétique de classe, puis de nom et de prénom de l'enseignant.

Indice :

> *On fera l'hypothèse que tous les étudiants d'une classe ont le même nombre de notes (pas d'absence aux examens).*

4. Quiz : SQL LMD

Exercice 1

Parmi les assertions suivantes concernant le langage SQL, sélectionner celles qui sont correctes.

☐	Le langage SQL permet d'implémenter physiquement un modèle logique exprimé en relationnel.
☐	Le langage SQL permet d'entrer et de sortir des données d'une base de données.
☐	Le langage SQL permet de donner et d'enlever des droits en lecture sur les données d'une base de données.
☐	Le langage SQL permet de donner et d'enlever des droits en écriture sur les données d'une base de données.
☐	Le langage SQL permet de créer une interface graphique utilisable par les utilisateurs finaux.
☐	Le langage SQL permet de faire des traitements algorithmiques complexes.
☐	Le langage SQL permet en général de créer une application informatique complète.

Exercice 2

Soit les deux relations suivantes UV1 et UV2 représentant respectivement les places disponibles dans les cours de l'UTC et les inscriptions effectives pour les cours ouverts ce semestre :

Nom	Nombre
NF17	168
NF18	48
NF19	48
NF20	48
NF21	48
NF22	168

UV1 : Place disponibles

Nom	Nombre
NF17	182
NF18	46
NF19	44
NF22	98

UV2 : Inscriptions

Tableau 4 Relations UV1 et UV2

*Compléter la requête SQL suivante pour qu'elle retourne pour **tous** les cours avec leur nom, plus le nombre d'inscrits en excès ou en défaut pour les cours ouverts et NULL pour les cours fermés.*

SELECT UV1.Nom, _____

FROM UV1 _____ UV2

ON _____

Exercice 3

Soit les deux relations suivantes UV1 et UV2 représentant respectivement les places disponibles dans les cours de l'UTC et les inscriptions effectives pour les cours ouverts ce semestre :

Nom	Nombre
NF17	168
NF18	48
NF19	48
NF20	48
NF21	48
NF22	168

UV1 : Place disponibles

Nom	Nombre
NF17	182
NF18	46
NF19	44
NF22	98

UV2 : Inscriptions

Tableau 5 Relations UV1 et UV2

Compléter la requête SQL suivante pour qu'elle retourne, pour les cours ouverts uniquement, le nombre d'inscrits moyen selon le nombre de places (nombre d'inscrits moyens pour les cours de 168 places, pour ceux de 48 places, etc.)

SELECT UV1.Nombre,
FROM UV1 UV2
ON

Exercice 4

Quelle sont les requêtes équivalente à la requête ci-dessous :

```
1  SELECT Nom
2  FROM Personne
3  WHERE NOT(Nom IS NULL)
4  GROUP BY Nom
```

- ☐
```
SELECT DISTINCT Nom
FROM Personne
WHERE Nom LIKE '%'
```
- ☐
```
SELECT Nom
FROM Personne
```
- ☐
```
SELECT Nom
FROM Personne
GROUP BY Nom
HAVING Count(Nom) > 0
```
- ☐
```
SELECT Nom
FROM Personne
WHERE NOT(Nom IS NULL)
```

Exercice 5

Que renvoie la dernière instruction SQL de la séquence ci-dessous ?

```
1  CREATE TABLE R (a INTEGER, b INTEGER);
2  INSERT INTO R VALUES (1,1);
3  INSERT INTO R VALUES (1,2);
4  INSERT INTO R VALUES (3,3);
5  SELECT sum(R1.b) FROM R R1, R R2 WHERE R1.a=R2.a;
```

Vues et gestion des droits

VI

A. Cours

1. Notion de schéma externe et de vue

a) 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.

b) 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.

Rappel : La séparation du niveau logique et du niveau physique

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.

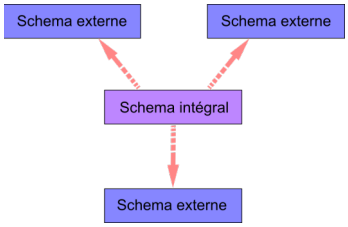
Notion de schéma externe

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.

c) Création de vues en SQL (CREATE VIEW)

Définition : Vue

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

```
1  CREATE VIEW <nom de vue> <nom des colonnes>
2  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

```
1  CREATE VIEW Employe (Id, Nom)
2  AS
3    SELECT N°SS, Nom
4    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.

Remarque : Vue en lecture et vue en écriture

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.

Remarque : Vue sur une vue

Une vue peut avoir comme source une autre vue.

Rappel : Vues et héritage

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.

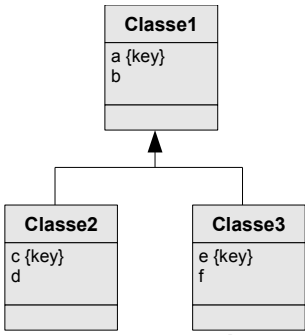
a) Héritage par une référence et vues

Rappel

Transformation de la relation d'héritage par référence

Méthode

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



Graphique 44 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)
```

Complément : Algèbre relationnel

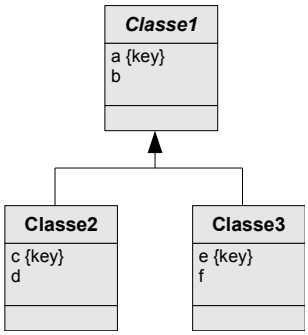
Jointure

b) Héritage par les classes filles et vues

Rappel

Transformation de la relation d'héritage par les classes filles

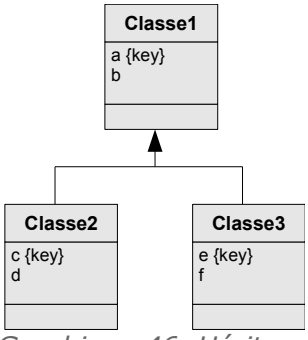
Méthode : Héritage absorbé par les classes filles (classe mère abstraite)



Graphique 45 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))
```

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



Graphique 46 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))
```

Exemple : Héritage absorbé par les classes filles

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))
```

Complément : Algèbre relationnel

Opérateurs ensemblistes

Projection

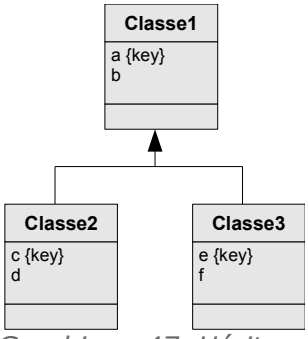
c) Héritage par la classe mère et vues

Rappel

Transformation de la relation d'héritage par la classe mère

Méthode

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



Graphique 47 Héritage

```
Classe1(#a,b,c,d,e,f,t:{1,2,3,23}) 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)
```

Exemple : Héritage absorbé par la classe mère

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 :

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

Complément : Algèbre relationnel

Projection

Restriction

d) 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.

Remarque : Attributs dérivés

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

e) 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 ».

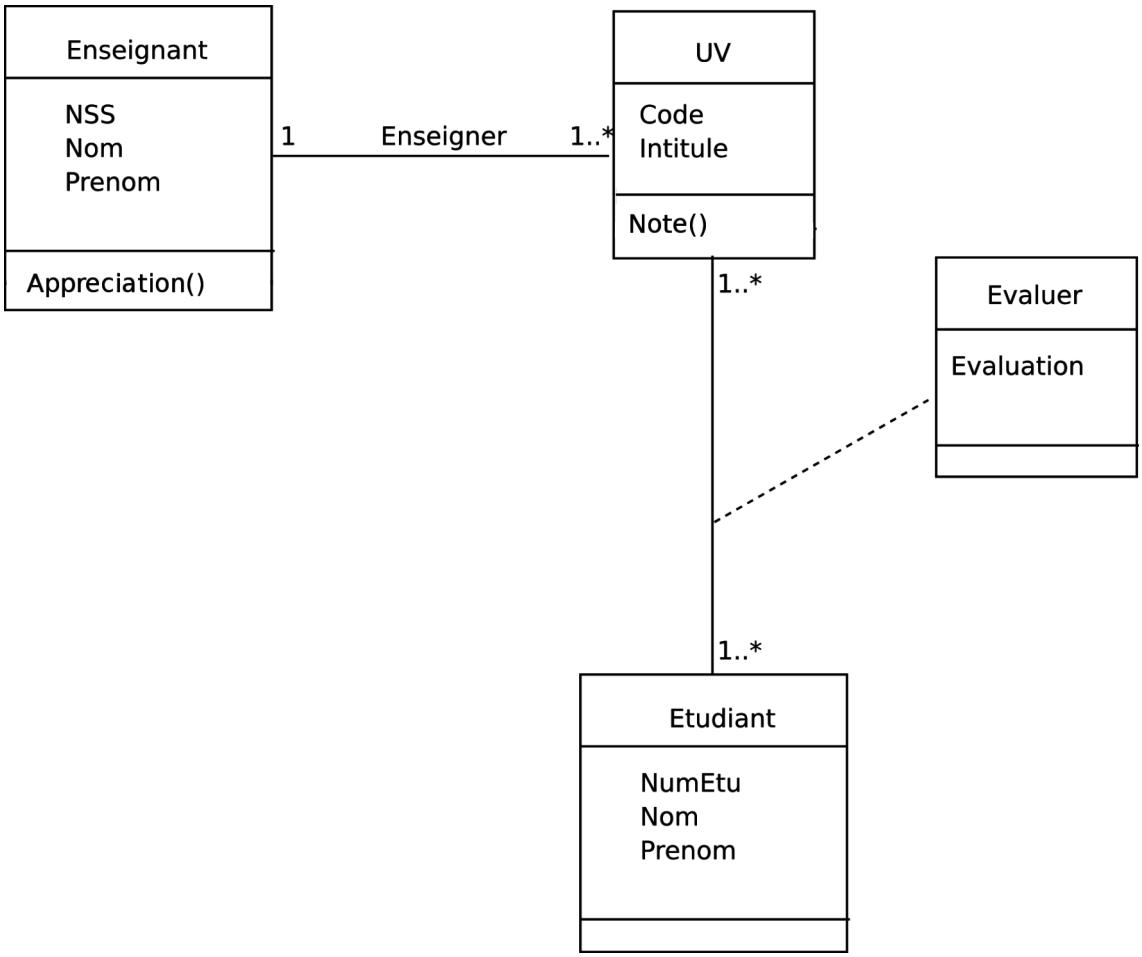


Image 27 Diagramme de classe

```
1  Enseignant(NSS, Nom, Prenom) ;
2  UV(Code, Intitulé, Prof=>Enseignant) ;
3  Etudiant(NumEtu, Nom, Prenom) ;
4  Evaluer(NumEtu=>Etudiant, uv=>UV, Evaluation) ;
```

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

Tableau 6 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.

a) 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 ?

```
1  REVOKE ALL PRIVILEGES ON * FROM Lambda;
2  GRANT UPDATE, SELECT ON a TO Lambda;
3  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')
```

b) Attribution de droits

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

Syntaxe

```
1 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

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

Remarque : Droits sur une vue

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).

Remarque : Catalogue de données

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.

Remarque : Création des utilisateurs

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.

c) Révocation de droits

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

Syntaxe

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

Exemple

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

Remarque : Révocation du droit de donner les droits

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.

Remarque : Révocation en cascade

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.

d) 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, ...).

e) The show must go on

[10 minutes]

Soit la base de données suivantes :

```
1  SPECTACLE (#nospectacle:int, nom:str, durée:int, type:{théâtre|danse|concert})
2  SALLE (#nosalle:int, nbplaces:int)
3  REPRESENTATION (#date:date, #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.

B. Exercice

1. Du producteur au consommateur++

[30 min]

Soit la base de données suivante :

```
1  CREATE TABLE  Producteur (
2  raison_sociale VARCHAR (25),
3  ville VARCHAR(255),
4  PRIMARY KEY (raison_sociale)
5  );
6
7  CREATE TABLE Consommateur (
8  login VARCHAR(10),
9  email VARCHAR(50),
```

```
10  nom VARCHAR(50) NOT NULL,
11  prenom VARCHAR(50) NOT NULL,
12  ville VARCHAR(255) NOT NULL,
13  PRIMARY KEY (login,email),
14  UNIQUE (nom,prenom,ville)
15  );
16
17  CREATE TABLE Produit (
18  id INTEGER,
19  description VARCHAR(100),
20  produit_par VARCHAR(25) NOT NULL,
21  consomme_par_login VARCHAR(10),
22  consomme_par_email VARCHAR(50),
23  PRIMARY KEY (id),
24  FOREIGN KEY (produit_par) REFERENCES Producteur(raison_sociale),
25  FOREIGN KEY (consomme_par_login,consomme_par_email) REFERENCES
     Consommateur(login,email)
26  );
```

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. 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 :

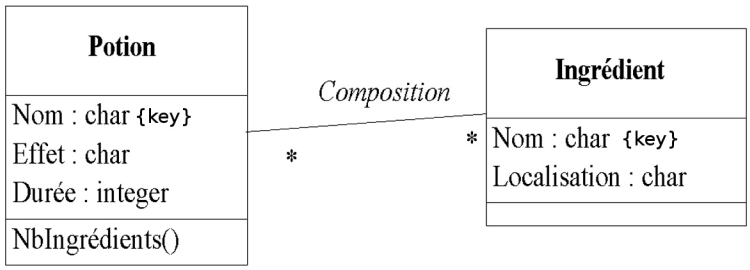


Image 28 Potion : modèle UML

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

```
1  /**
2  DROP TABLE Composition ;
3  DROP TABLE Ingredient ;
4  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.

Théorie de la normalisation relationnelle

VII

A. Cours

La théorie de la normalisation relationnelle est très importante pour la conception de BD★, dans la mesure où elle donne le cadre théorique pour la gestion de la redondance, et dans la mesure où une bonne maîtrise de la redondance est un aspect majeur de cette conception.

1. Redondance et normalisation

Objectifs

Comprendre la problématique de la redondance.

a) Introduction à la redondance

Soit la relation R suivante, définie en extension :

A	B	C	D	E	F	G
0	1	1	10	5	X	A
0	2	1	10	9	X	G
0	1	2	10	6	X	S
0	1	3	10	7	X	D
1	2	3	20	7	Y	D
0	3	3	10	9	X	G
1	4	3	20	8	Y	F
1	1	4	20	9	Y	G

Tableau 7 Relation R

Question 1

Proposez des clés pour cette relation. Justifiez brièvement.

Question 2

Cette relation contient-elle des redondances ? Si oui lesquelles ? Justifiez brièvement.

Question 3

Si la relation contient des redondances, proposez une solution contenant exactement la même information, mais sans redondance.

b) Les problèmes soulevés par une mauvaise modélisation

Attention

Il y a toujours plusieurs façons de modéliser conceptuellement un problème, certaines sont bonnes et d'autres mauvaises. C'est l'expertise de l'ingénieur en charge de la modélisation, à travers son expérience accumulée et sa capacité à traduire le problème posé, qui permet d'obtenir de bons modèles conceptuels.

S'il est difficile de définir un bon modèle conceptuel, on peut en revanche poser qu'un bon modèle logique relationnel est un modèle où la redondance est contrôlée.

On peut alors poser qu'un bon modèle conceptuel est un modèle conceptuel qui conduit à un bon modèle relationnel, après application des règles de passage E-A ou UML vers relationnel. Mais on ne sait pas pour autant le critiquer avant ce passage, autrement qu'à travers l'œil d'un expert.

A défaut de disposer d'outils systématiques pour obtenir de bons modèles conceptuels, on cherche donc à critiquer les modèles relationnels obtenus.

La théorie de la normalisation est une théorie qui permet de critiquer, puis d'optimiser, des modèles relationnels, de façon à en contrôler la redondance.

Exemple : Un mauvais modèle relationnel

Imaginons que nous souhaitions représenter des personnes, identifiées par leur numéro de sécurité sociale, caractérisées par leur nom, leur prénom, ainsi que les véhicule qu'elles ont acheté, pour un certain prix et à une certaine date, sachant qu'un véhicule est caractérisé par son numéro d'immatriculation, un type, une marque et une puissance. On peut aboutir à la représentation relationnelle suivante :

```
1  Personne(NSS, Nom, Prénom, Immat, Marque, Type, Puiss, Date, Prix)
```

Posons que cette relation soit remplie par les données suivantes :

NSS	Nom	Prénom	Immat	Marque	Type	Puiss	Date	Prix
16607...	Dupont	Paul	AJ600AQ	Renault	Clio	5	1/1/96	60000
16607...	Dupont	Paul	AA751KK	Peugeot	504	7	2/7/75	47300
24908...	Martin	Marie	AA751KK	Peugeot	504	7	1/10/89	54900
15405...	Durand	Olivier	AA751KK	Peugeot	504	7	8/8/90	12000
15405...	Durand	Olivier	AJ600AQ	Renault	Clio	5	7/6/98	65000
15405...	Durand	Olivier	XX100XX	BMW	520	10	4/5/01	98000

Tableau 8 Relation redondante

On peut alors se rendre compte que des redondances sont présentes, si l'on connaît NSS on connaît Nom et Prénom, si on connaît Immat, on connaît Marque, Type et Puiss.

NSS	Nom	Prénom	Immat	Marque	Type	Puiss	Date	Prix
16607...	Dupont	Paul	AJ600AQ	Renault	Clio	5	1/1/96	60000
16607...	Dupont	Paul	AA751KK	Peugeot	504	7	2/7/75	47300
24908...	Martin	Marie	AA751KK	Peugeot	504	7	1/10/89	54900
15405...	Durand	Olivier	AA751KK	Peugeot	504	7	8/8/90	12000
15405...	Durand	Olivier	AJ600AQ	Renault	Clio	5	7/6/98	65000
15405...	Durand	Olivier	XX100XX	BMW	520	10	4/5/01	98000

Relation redondante

On sait que ces redondances conduiront à des problèmes de contrôle de la cohérence de l'information (erreur dans la saisie d'un numéro de sécurité sociale), de mise à jour (changement de nom à reporter dans de multiples tuples), de perte d'information lors de la suppression de données (disparition des informations concernant un type de véhicule) et de difficulté à représenter certaines informations (un type de véhicule sans propriétaire).

Complément

On conseillera de lire le chapitre 2 de SQL2 SQL3, applications à Oracle [Delmal01] (pages 42 à 49) qui propose une très bonne démonstration par l'exemple des problèmes posés par une mauvaise modélisation relationnelle.

c) Principes de la normalisation

Fondamental

La théorie de la normalisation est une théorie destinée à concevoir un bon schéma d'une base de données sans redondance d'information et sans risques d'anomalie de mise à jour. Elle a été introduite dès l'origine dans le modèle relationnel.

La théorie de la normalisation est fondée sur deux concepts principaux :

- **Les dépendances fonctionnelles**

  Elles traduisent des contraintes sur les données.
- **Les formes normales**

  Elles définissent des relations bien conçues.

La mise en œuvre de la normalisation est fondée sur la décomposition progressive des relations jusqu'à obtenir des relations normalisées.

2. Les dépendances fonctionnelles

Objectifs

Savoir repérer et exprimer des dépendances fonctionnelles.
Définir une clé par les dépendances fonctionnelles.

a) Exercice : A1, dans l'eau !

Considérons le schéma de la relation suivante :

- `r (A, B, C, D, E)`

Cette relation est définie en intension par les tuples suivants :

A	B	C	D	E
a1	b2	c2	d3	e2
a1	b2	c2	d1	e4
a2	b3	c2	d1	e5
a2	b4	c5	d1	e5

Parmi les dépendances fonctionnelles suivantes, lesquelles s'appliquent à r ?

- ☐ E→D
- ☐ D→E
- ☐ C→A
- ☐ E→B
- ☐ E→A
- ☐ B→C
- ☐ B→D
- ☐ B→A

b) Dépendance fonctionnelle

Définition : Dépendance fonctionnelle

Soient R(A1, A2, ... , An) un schéma de relation, X et Y des sous-ensembles de A1, A2, ... , An. On dit que X détermine Y, ou que Y dépend fonctionnellement de X, si et seulement s'il existe une fonction qui à partir de toute valeur de X détermine une valeur unique de Y.

Plus formellement on pose que X détermine Y pour une relation R si et seulement si quelle que soit l'instance r de R, alors pour tous tuples t1 et t2 de r on a :

Projection (t1,X) = Projection (t2,X) ⇒ Projection (t1,Y) = Projection (t2,Y)

Syntaxe

Si X détermine Y, on note : X→Y

Exemple

Soit la relation R suivante :

```
1 Personne(NSS, Nom, Prénom, Marque, Type, Puiss, Date, Prix)
```

On peut poser les exemples de DF★ suivants :

- NSS→Nom

- NSS→Prénom
- Type→Marque
- Type→Puiss
- (NSS, Type, Date)→Prix
- etc.

Remarque : Comment trouver les DF ?

Une DF est définie sur l'intension du schéma et non son extension. Une DF traduit une certaine perception de la réalité. Ainsi la DF (NSS, Type, Date)→Prix signifie que personne n'achète deux voitures du même type à la même date.

La seule manière de déterminer une DF est donc de regarder soigneusement ce que signifient les attributs et de trouver les contraintes qui les lient dans le monde réel.

Remarque : Pourquoi trouver les DF ?

Les DF font partie du schéma d'une BD, en conséquence, elles doivent être déclarées par les administrateurs de la BD et être contrôlées par le SGBD.

De plus l'identification des DF est la base indispensable pour déterminer dans quelle forme normale est une relation et comment en diminuer la redondance.

c) Les axiomes d'Armstrong

Introduction

Les DF✯ obéissent à des propriétés mathématiques particulières, dites axiomes d'Armstrong.

Définition : Réflexivité

Tout groupe d'attributs se détermine lui même et détermine chacun de ses attributs (ou sous groupe de ses attributs).

Soient X et Y des attributs :

XY→XY et XY→X et XY→Y

Définition : Augmentation

Si un attribut X détermine un attribut Y, alors tout groupe composé de X enrichi avec d'autres attributs détermine un groupe composé de Y et enrichi des mêmes autres attributs.

Soient X, Y et Z des attributs :

X→Y ⇒ XZ→YZ

Définition : Transitivité

Si un attribut X détermine un attribut Y et que cet attribut Y détermine un autre attribut Z, alors X détermine Z.

Soient X, Y et Z des attributs :

X→Y et Y→Z ⇒ X→Z

d) Autres propriétés déduites des axiomes d'Armstrong

Introduction

A partir des axiomes d'Amstrong, on peut déduire un certain nombre de propriétés supplémentaires.

Définition : Pseudo-transitivité

Si un attribut X détermine un autre attribut Y, et que Y appartient à un groupe G qui détermine un troisième attribut Z, alors le groupe G' obtenu en substituant Y par X dans G détermine également Z.

Soient, W, X, Y et Z des attributs :

X→Y et WY→Z ⇒ WX→Z

Cette propriété est déduite de l'augmentation et de la réflexivité :

X→Y et WY→Z ⇒ WX→WY et WY→Z ⇒ WX→Z

Définition : Union

Si un attribut détermine plusieurs autres attributs, alors il détermine tout groupe composé de ces attributs.

Soient X, Y et Z des attributs :

X→Y et X→Z ⇒ X→YZ

Cette propriété est déduite de la réflexivité, de l'augmentation et de la transitivité :

X→Y et X→Z ⇒ X→XX et XX→XY et YX→YZ ⇒ X→YZ

Définition : Décomposition

Si un attribut détermine un groupe d'attribut, alors il détermine chacun des attributs de ce groupe pris individuellement.

Soient X, Y et Z des attributs :

X→YZ ⇒ X→Z et X→Y

Cette propriété est déduite de la réflexivité et de la transitivité :

X→YZ ⇒ X→YZ et YZ→Z ⇒ X→Z

e) DF élémentaire

Définition : Dépendance fonctionnelle élémentaire

Soit G un groupe d'attributs et A un attribut, une DF✯ G→A est élémentaire si A n'est pas incluse dans G et s'il n'existe pas d'attribut A' de G qui détermine A.

Exemple : DF élémentaires

- AB→C est élémentaire si ni A, ni B pris individuellement ne déterminent C.
- Nom, DateNaissance, LieuNaissance→Prénom est élémentaire.

Exemple : DF non élémentaires

- AB→A n'est pas élémentaire car A est incluse dans AB.
- AB→CB n'est pas élémentaire car CB n'est pas un attribut, mais un groupe d'attributs.
- N°SS→Nom, Prénom n'est pas élémentaire.

Remarque

On peut toujours réécrire un ensemble de DF en un ensemble de DFE✯, en supprimant les DF triviales obtenues par réflexivité et en décomposant les DF à partie droite non atomique en plusieurs DFE.

Exemple : Réécriture de DF en DFE

On peut réécrire les DF non élémentaires de l'exemple précédent en les décomposant DFE :

- AB→A n'est pas considérée car c'est une DF triviale obtenu par réflexivité.
- AB→CB est décomposée en AB→C et AB→B, et AB→B n'est plus considérée car triviale.
- N°SS→Nom, Prénom est décomposée en N°SS→Nom et N°SS→Prénom.

f) Notion de fermeture transitive des DFE

Définition : Fermeture transitive

On appelle fermeture transitive F+ d'un ensemble F de DFE✯, l'ensemble de toutes les DFE qui peuvent être composées par transitivité à partir des DFE de F.

Exemple

Soit l'ensemble F = {A→B, B→C, B→D, A→E}.

La fermeture transitive de F est F+ = { A→B, B→C, B→D, A→E, A→C, A→D }

g) Notion de couverture minimale des DFE

Définition : Couverture minimale

La couverture minimale d'un ensemble de DFE✯ est un sous-ensemble minimum des DFE permettant de générer toutes les autres DFE.

Synonymes : Famille génératrice

Remarque

Tout ensemble de DFE (et donc tout ensemble de DF) admet au moins une couverture minimale (et en pratique souvent plusieurs).

Exemple

L'ensemble F = {A→B, A→C, B→C, C→B} admet les deux couvertures minimales :

CM1 = {A→C, B→C, C→B} et CM2 = {A→B, B→C, C→B}

h) Notion de graphe des DFE

On peut représenter un ensemble de DFE par un graphe orienté (ou plus précisément un réseau de Pétri), tel que les nœuds sont les attributs et les arcs les DFE (avec un seul attribut en destination de chaque arc et éventuellement plusieurs en source).

Exemple : Relation Voiture

Soit la relation Voiture(NVH, Marque, Type, Puis, Couleur) avec l'ensemble des DF F = {NVH→Type, Type→Marque, Type→Puis, NVH→Couleur}. On peut représenter F par le graphe ci-dessous :

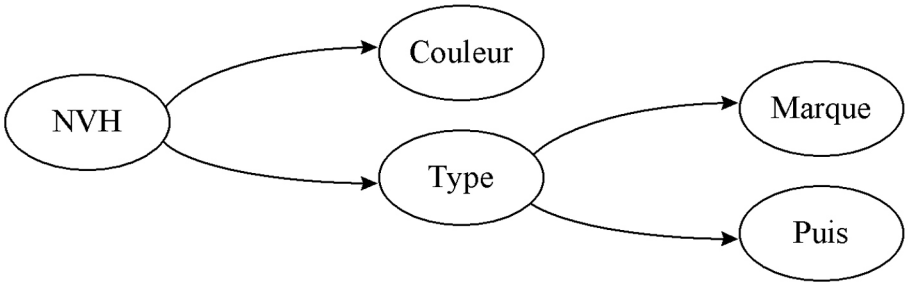


Image 29 Graphe des DFE de la relation Voiture

Exemple : Relation CodePostal

Soit la relation CodePostal(Code, Ville, Rue) avec l'ensemble des DF F={Code→Ville, (Ville,Rue)→Code}. On peut réprésenter F par le graphe ci-dessous :

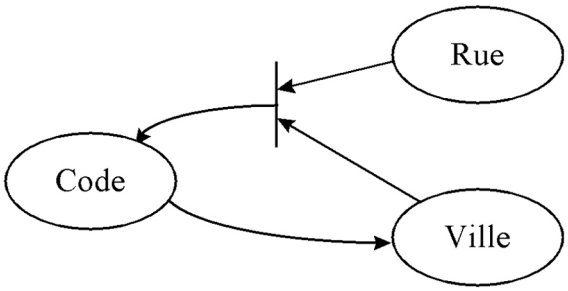


Image 30 Graphe des DFE de la relation CodePostal

i) Définition formelle d'une clé

Définition : Clé

Soient une relation R(A1,A2,...,An) et K un sous-ensemble de A1,A2,... ,An.

K est une clé de R si et seulement si :

1. K→A1,A2,...,An
2. et il n'existe pas X inclus dans K tel que X→A1,A2,...,An.

Fondamental

Une clé est donc un ensemble minimum d'attributs d'une relation qui détermine tous les autres.

Remarque : Clés candidates et clé primaire

Si une relation comporte plusieurs clés, chacune est dite clé candidate et l'on en choisit une en particulier pour être la clé primaire.

Attention : Les clés candidates sont des clés !

Toutes les clés candidates sont des clés, pas seulement la clé primaire.

Remarque : Les clés candidates se déterminent mutuellement

Toute clé candidate détermine les autres clés candidates, puisque qu'une clé détermine tous les attributs de la relation.

Complément : Relation "toute clé"

Étant donné qu'une relation dispose forcément d'une clé, si une relation R n'admet aucune clé K sous ensemble des attributs A1..An de R, alors c'est que K=A1..An (la clé est composée de **tous** les attributs de R).

On parle de relation "toute clé".

j) Exercice : A1, touché !

Considérons le schéma de la relation suivante :

- `r (A, B, C, D, E)`

Cette relation est définie en intension par les tuples suivants :

A	B	C	D	E
a1	b2	c2	d3	e2
a1	b2	c2	d1	e4
a2	b3	c2	d1	e5
a2	b4	c5	d1	e5

Après avoir énoncé les DF, déterminer, parmi les groupes d'attributs suivants, lesquels sont des clés ?

- ☐ A
- ☐ B
- ☐ C
- ☐ D
- ☐ E
- ☐ {B,E}
- ☐ {A,B,C,D,E}

3. Les formes normales

Objectifs

Connaître les formes normales et leurs implications en terme de redondance.

a) Formes normales

Les formes normales ont pour objectif de définir la décomposition des schémas relationnels, tout en préservant les DF★ et sans perdre d'informations, afin de représenter les objets et associations canoniques du monde réel de façon non redondante.

On peut recenser les 6 formes normales suivantes, de moins en moins redondantes :

- la première forme normale
- la deuxième forme normale
- la troisième forme normale
- la forme normale de Boyce-Codd
- la quatrième forme normale
- la cinquième forme normale

La troisième forme normale est généralement reconnue comme étant la plus importante à respecter.

b) Principe de la décomposition

L'objectif de la décomposition est de "casser" une relation en relations plus petites afin d'en éliminer les redondances et sans perdre d'information.

Définition : Décomposition

La décomposition d'un schéma de relation R(A1,A2,...,An) est le processus de remplacement de ce schéma par une collection de schémas R1,R2,...,Rn telle qu'il est possible de reconstruire R par des opérations relationnelles de jointure sur R1,R2,...,Rn.

Définition : Décomposition préservant les DF

Une décomposition d'une relation R en relations R1,R2,...Rn préserve les DF✯ si la fermeture transitive F+ des DF de R est la même que celle de l'union des fermetures transitives des DF de R1,R2,...,Rn.

Exemple : Décomposition préservant les DF d'une relation Voiture

Soit la relation Voiture(Numéro,Marque,Type,Puissance,Couleur) avec la fermeture transitive suivante :

- Numéro→Marque
- Numéro→Type
- Numéro→Puissance
- Numéro→Couleur
- Type→Marque
- Type→Puissance

On peut décomposer Voiture en préservant les DF en deux relations R1(Numéro,Type,Couleur) et R2(Type,Puissance,Marque).

c) Première forme normale

Définition : 1NF

Une relation est en 1NF✯ si elle possède au moins une clé et si tous ses attributs sont atomiques.

Définition : Attribut atomique

Un attribut est atomique si il ne contient qu'une seule valeur pour un tuple donné, et donc s'il ne regroupe pas un ensemble de plusieurs valeurs.

Exemple : Avoir plusieurs métiers

Soit la relation Personne instanciée par deux tuples :

```
1  Personne(#Nom, Profession)
```

```
1  (Dupont, Géomètre)
2  (Durand, Ingénieur-Professeur)
```

La relation n'est pas en 1NF, car l'attribut Profession peut contenir plusieurs valeurs.

Pour que la relation soit en 1NF, on pourrait par exemple ajouter Profession à la clé et faire apparaître deux tuples pour Durand, on obtiendrait :

```
1  Personne(#Nom, #Profession)
```

```
1 (Dupont, Géomètre)
2 (Durand, Ingénieur)
3 (Durand, Professeur)
```

Une autre solution aurait été d'ajouter un attribut ProfessionSecondaire. On obtiendrait ainsi :

```
1 Personne(#Nom, Profession, ProfessionSecondaire)
```

```
1 (Dupont, Géomètre, Null)
2 (Durand, Ingénieur, Professeur)
```

Remarque : Relativité de la notion d'atomicité

L'atomicité d'un attribut est souvent relative : on peut décider qu'un attribut contenant une date n'est pas atomique (et que le jour, le mois et l'année constituent chacun une valeur), ou bien que l'attribut est de domaine date et donc qu'il est atomique.

Fondamental : Énoncer les clés

Le modèle relationnel impose qu'une relation ait une clé, donc la condition "est en 1NF si elle possède une clé" est superflue (au pire la relation est toute clé).

Il est néanmoins fondamental d'avoir identifié les clés au début du processus de normalisation.

d) Deuxième forme normale

Introduction

La deuxième forme normale permet d'éliminer les dépendances entre des parties de clé et des attributs n'appartenant pas à une clé.

Définition : 2NF

Une relation est en 2NF★ si elle est en 1NF★ et si tout attribut qui n'est pas dans une clé ne dépend pas d'une partie seulement d'une clé. C'est à dire encore que toutes les DF★ issues d'une clé sont élémentaires.

Exemple : Echelle de salaire

Soit la relation Personne :

```
1 Personne(#Nom, #Profession, Salaire)
```

Soit les DF suivantes sur cette relation :

- Nom,Profession→Salaire
- Profession→Salaire

On note alors que la première DF est issue de la clé et qu'elle n'est pas élémentaire (puisque Profession détermine Salaire) et donc que le schéma n'est pas en 2NF.

Pour avoir un schéma relationnel en 2NF, il faut alors décomposer Personne en deux relations :

```
1 Personne(#Nom, #Profession=>Profession, Prenom)
2 Profession(#Profession, Salaire)
```

On remarque que ce schéma est en 2NF (puisque Salaire dépend maintenant fonctionnellement d'une clé et non plus d'une partie de clé).

On remarque aussi que la décomposition a préservé les DF, puisque nous avons à présent :

- Profession→Salaire (DF de la relation Profession)
- Nom,Profession→Profession (par Réflexivité)

- Nom,Profession→Salaire (par Transitivité)

Attention

La définition de la 2NF doit être vérifiée pour toutes les clés candidates et non seulement la clé primaire (dans le cas où il y a plusieurs clés).

Remarque

Si toutes les clés d'une relation ne contiennent qu'un unique attribut, et que la relation est en 1NF, alors la relation est en 2NF.

e) Troisième forme normale

Introduction

La troisième forme normale permet d'éliminer les dépendances entre les attributs n'appartenant pas à une clé.

Définition : 3NF

Une relation est en 3NF✯ si elle est en 2NF✯ et si tout attribut n'appartenant pas à une clé ne dépend pas d'un autre attribut n'appartenant pas à une clé. C'est à dire encore que toutes les DFE✯ vers des attributs n'appartenant pas à une clé, sont issues d'une clé.

Attention : Clé candidate

La définition concerne toutes les clés candidates et non uniquement la clé primaire (SQL avancé : Programmation et techniques avancées [Celko00], p.27).

Exemple : Échelle de salaire et de prime

Soit la relation Profession :

```
1  Profession(#Profession, Salaire, Prime)
```

Soit les DF suivantes sur cette relation :

- Profession→Salaire
- Profession→Prime
- Salaire→Prime

Cette relation n'est pas en 3NF car Salaire, qui n'est pas une clé, détermine Prime.

Pour avoir un schéma relationnel en 3NF, il faut décomposer Profession :

```
1  Profession(#Profession, Salaire=>Salaire)
2  Salaire(#Salaire, Prime)
```

Ce schéma est en 3NF, car Prime est maintenant déterminé par une clé.

On remarque que cette décomposition préserve les DF, car par transitivité, Profession détermine Salaire qui détermine Prime, et donc Profession détermine toujours Prime.

Remarque : 3NF et 2NF

Une relation en 3NF est forcément en 2NF car :

- Toutes les DFE vers des attributs n'appartenant pas à une clé sont issues d'une clé, ce qui implique qu'il n'existe pas de DFE, issues d'une partie de clé vers un attribut qui n'appartient pas à une clé.
- Il ne peut pas non plus exister de DFE issues d'une partie de clé vers un attribut appartenant à une clé, par définition de ce qu'une clé est un ensemble minimum.

On n'en conclut qu'il ne peut exister de DFE, donc a fortiori pas de DF✯, issues d'une partie d'une clé, et donc que toutes les DF issues d'une clé sont élémentaires.

Fondamental

Il est souhaitable que les relations logiques soient en 3NF. En effet, il existe toujours une décomposition sans perte d'information et préservant les DF d'un schéma en 3NF. Si les formes normales suivantes (BCNF★, 4NF★ et 5NF★) assurent un niveau de redondance encore plus faible, la décomposition permettant de les atteindre ne préserve plus les DF.

Remarque : Limite de la 3NF

Une relation en 3NF permet des dépendances entre des attributs n'appartenant pas à une clé vers des parties de clé.

f) Forme normale de Boyce-Codd

Introduction

La forme normale de Boyce-Codd permet d'éliminer les dépendances entre les attributs n'appartenant pas à une clé vers les parties de clé.

Définition : BCNF

Une relation est en BCNF★ si elle est en 3NF★ et si tout attribut qui n'appartient pas à une clé n'est pas source d'une DF★ vers une partie d'une clé. C'est à dire que les seules DFE★ existantes sont celles dans lesquelles une clé détermine un attribut.

Exemple : Employés

Soit la relation Personne :

```
1  Personne(#N°SS, #Pays, Nom, Région)
```

Soit les DF suivantes sur cette relation :

- N°SS,Pays→Nom
  N°SS,Pays→Région
  Région→Pays

Il existe une DFE qui n'est pas issue d'une clé et qui détermine un attribut appartenant à une clé. Cette relation est en 3NF, mais pas en BCNF (car en BCNF toutes les DFE sont issues d'une clé).

Pour avoir un schéma relationnel en BCNF, il faut décomposer Personne :

```
1  Personne(#N°SS, #Region=>Region, Nom)
2  Region(#Region, Pays)
```

Remarquons que les DF n'ont pas été préservées par la décomposition puisque N°SS et Pays ne déterminent plus Région.

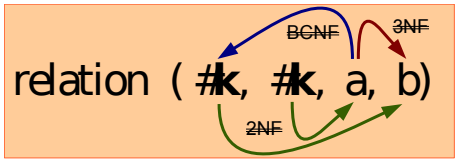
Remarque : Simplicité

La BCNF est la forme normale la plus facile à appréhender intuitivement et formellement, puisque les seules DFE existantes sont de la forme K→A où K est une clé.

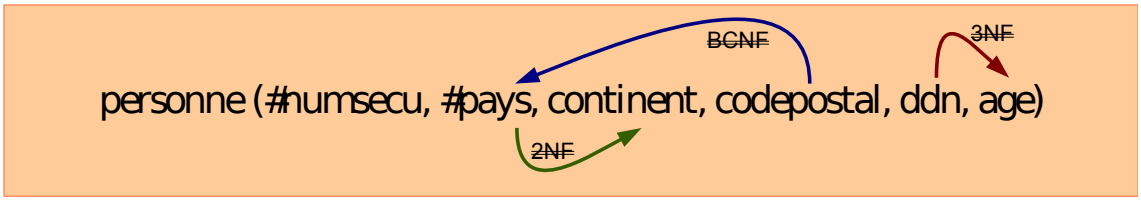
Attention : Non préservation des DF

Une décomposition en BCNF ne préserve pas toujours les DF.

g) Synthèse



Exemple



h) Exercice : A1, coulé !

Considérons le schéma de la relation suivante :

- `r (A, B, C, D, E)`

Cette relation est définie en intension par les tuples suivants :

A	B	C	D	E
a1	b2	c2	d3	e2
a1	b2	c2	d1	e4
a2	b3	c2	d1	e5
a2	b4	c5	d1	e5

Après avoir énoncé les DF et les clés, déterminer la forme normale du schéma ?

- ☐ 1NF
- ☐ 2NF
- ☐ 3NF
- ☐ BCNF

4. Bibliographie commentée sur la normalisation

Complément : Synthèses

SQL2 SQL3, applications à Oracle [Delmal01]

On conseillera de lire le chapitre 2 (pages 42 à 49) qui propose une très bonne démonstration par l'exemple des problèmes posés par une mauvaise modélisation relationnelle.

Une description claire des formes normales, rendue simple et pratique grâce à des exemples représentatifs (chapitre 2).

B. Exercices

1. De quoi dépend un cours ?

[10 min]

On considère le schéma relationnel R défini sur les attributs suivants : C : cours, P : professeur, H : heure, S : salle, E : étudiant, N : note.

Un nuplet (c, p, h, s, e, n) a pour signification que le cours c est fait par le professeur p à l'heure h dans la salle s pour l'étudiant e qui a reçu la note n.

L'ensemble E des dépendances fonctionnelles initiales est le suivant :

- C → P
- H,S → C
- H,P → S
- C,E → N
- H,E → S

Question 1

Donner la fermeture transitive F+ des dépendances fonctionnelles élémentaires engendrées par E.

Question 2

Quelle est la clé de la relation R ? Montrer qu'elle est unique.

2. Cuisines et dépendances

[20 min]

On considère une relation R construite sur les attributs Propriétaire, Occupant, Adresse, Noapt, Nbpièces, Nbpersonnes, un nuplet (p, o, a, n, nb1, nb2) ayant la signification suivante : La personne o habite avec nb2 personnes l'appartement de numéro n ayant nb1 pièces dont le propriétaire est p.

Une analyse de cette relation nous fournit un ensemble initial E de dépendances fonctionnelles :

- occupant → adresse
- occupant → noapt
- occupant → nbpersonnes
- adresse, noapt → propriétaire
- adresse, noapt → occupant
- adresse, noapt → nbpièces

Question 1

Donner l'ensemble des DFE engendrées par E (fermeture transitive F+).

Question 2

Quelles sont les clés candidates de R ?

Question 3

Montrer que R est en 3NF de deux façons différentes (sans passer et en passant par la BCNF).

3. Test : Normalisation

Exercice 1

Soit la relation suivante et une couverture minimale des DF associée.

```
1 tUtilisateur (pklogin,mdp,nom,prenom,ville)
```

- *pklogin→mdp,nom,prenom,ville*
- *nom,prenom→pklogin*
- *ville→nom*

Sélectionner la ou les clés de cette relation.

☐	pklogin
☐	mdp
☐	nom
☐	prenom
☐	ville
☐	(pklogin,mdp)
☐	(pklogin,nom)
☐	(nom,prenom)
☐	(ville,nom)
☐	(nom,prenom,ville)

Exercice 2

Soit le schéma relationnel (on pose que les attributs A, B, C, D, X et Y sont atomiques) :

```
1 R1(A, B, C, D)
2 R2(X, Y)
```

Soit les dépendances fonctionnelles identifiées :

- $A \rightarrow C$
- $A,B \rightarrow D$
- $X \rightarrow Y$

En quelles formes normales est ce schéma relationnel ?

☐	1NF
☐	2NF
☐	3NF
☐	BCNF

Exercice 3

Soit le schéma relationnel suivant :

```
1  Personne (Nom, Prenom, Age, DateNaissance)
```

Soit les DF suivantes :

- *Nom → Prenom*
- *Nom, Prenom → Age*
- *Prenom → DateNaissance, Nom*
- *DateNaissance, Age → Age, Nom*
- *Age, Nom → Age, DateNaissance*
- *DateNaissance → Age*

Quelles sont les clés candidates pour la relation Personne ?

☐	Nom
☐	Prenom
☐	Age
☐	DateNaissance

Exercice 4

Soit la relation R (A:Int, B:Int, C:Int, D:Int, E:Int) et l'ensemble de DFE✯ *suivant : {A → B ; A → C ; A → D ; A → E ; B → A ; B → C}*

Sélectionner toutes les assertions vraies.

☐	A est une clé
☐	B est une clé
☐	C est une clé
☐	Le schéma est en 1NF
☐	Le schéma est en 2 NF
☐	Le schéma est en 3 NF
☐	Cet ensemble de DFE est une fermeture transitive.
☐	Cet ensemble de DFE est une couverture minimale.

Exercice 5

Soit le schéma relationnel (on pose que tous les attributs sont atomiques) :

```
1  Adresse (Numero, Rue, Ville=>Ville, Pays=>Ville)
2  Ville (Ville, Pays=>Pays)
3  Pays (Pays)
```

En quelles formes normales est ce schéma relationnel ?

☐	1NF
☐	2NF
☐	3NF
☐	BCNF

Conception de bases de données normalisées

VIII

A. Cours

1. Conception de bases de données normalisées

Objectifs

Savoir créer des schémas relationnels en troisième forme normale.

a) Exercice : Étapes de la conception d'une base de données

Mettre dans l'ordre les étapes de conception suivantes.

1. Énonciation de la forme normale avant normalisation
2. Création du code SQL LDD pour un SGBDR ou un SGBDRO
3. Élaboration du modèle logique en relationnel ou relationnel-objet
4. Modélisation conceptuelle en UML ou E-A
5. Décomposition des relations jusqu'à arriver en 3NF en préservant les DF
6. Rétro-conception d'un modèle UML normalisé
7. Établissement des DF sous la forme de la fermeture transitive pour chaque relation du modèle
8. Analyse de la situation existante et des besoins

Réponse : ___ ___ ___ ___ ___ ___ ___ ___

b) Algorithme de décomposition

Méthode : Décomposition 0NF->1NF

Soit R une relation avec la clé primaire pk. Si R contient un attribut non atomique, alors R est décomposée en R1 et R2, tel que :

- R1 est R moins l'attribut a
- R2 est composé de pk et de a, avec (pk,a) clé primaire de R2 et R2.pk clé étrangère vers R1.pk

`R(#pk,a,b,...)` avec `a` non atomique se décompose en :

- `R1(#pk,b,...)`
- `R2(#pk=>R1,#a)`

Fondamental: Décomposition > 1NF

Pour les NF supérieures à 1, afin de normaliser une relation R on réalise une décomposition en R1 et R2 pour chaque DFE responsable d'un défaut de normalisation tel que :

- **la partie gauche de la DFE :**
  a. **devient la clé primaire de R2**
  b. **devient une clé étrangère de R1 vers R2**
- **la partie droite de la DFE**
  a. **est enlevée de R1**
  b. **est ajoutée comme attributs simples de R2**

Méthode : Décomposition 1NF->2NF

Soit R une relation comportant une clé composée de k1 et k1'. Si R contient une DFE de k1' vers des attributs n'appartenant pas à la clé, alors R est décomposée en R1 et R2, tel que :

- R1 est R moins les attributs déterminés par k1' et avec k1' clé étrangère vers R2
- R2 est composée de k1' et des attributs qu'elle détermine, avec k1' clé primaire de R2

`R(#pk,k1,k1',a,b,c,...)` avec (k1,K1') clé et k1'→a,b se décompose en :

- `R1(#pk,k1,k1'=>R2,c,...)`
- `R2(#k1',a,b)`

Méthode : Décomposition 2NF->3NF

Soit R une relation comportant une DFE de a vers b qui n'appartiennent pas à une clé, alors R est décomposée en R1 et R2, tel que :

- R1 est R moins les attributs déterminés par a et avec a clé étrangère vers R2
- R2 est composée de a et des attributs qu'elle détermine, avec a clé primaire de R2

`R(#pk,a,b,c,...)` avec a→b se décompose en

- `R1(#pk,a=>R2,c,...)`
- `R2(#a,b)`

c) Exemple de décomposition

Exemple : Situation initiale

```
1  Resultat (#pknum,knumetu,kuv,prenom,nom,credits,resultat,obtenu) avec
   (knumetu,kuv) clé
```

pknum	knumetu	kuv	prenom	nom	credits	resultat	obtenu
1	X01	NF17	Pierre	Alpha	6	A	oui
2	X01	NF26	Pierre	Alpha	6	B	oui
3	X02	NF17	Alphonse	Béta	6	F	non

- pknum → knumetu,kuv,prenom,nom,credits,resultat,obtenu
- knumetu,kuv → pknum,prenom,nom,credits,resultat,obtenu
- **knumetu → prenom,nom**
- **kuv → credits**

- **resultat → obtenu**

knumetu → prenom,nom

```
1 Resultat (#pknum,knumetu=>Etudiant,kuv,credits,resultat,obtenu)
2 Etudiant (#knumetu,prenom,nom)
```

kuv → credits

```
1 Resultat (#pknum,knumetu=>Etudiant,kuv=>Uv,resultat,obtenu)
2 Etudiant (#knumetu,prenom,nom)
3 Uv(#kuv,credits)
```

resultat → obtenu

```
1 Resultat (#pknum,knumetu=>Etudiant,kuv=>Uv,resultat=>Note)
2 Etudiant (#knumetu,prenom,nom)
3 Uv(#kuv,credits)
4 Note(#resultat,obtenu)
```

d) Normalisation par transformation d'attributs en méthodes

Il arrive que la fonction sous-jacente à la DF soit une fonction simple, que l'on peut calculer. Par exemple pour la DF : `ddn → age ( ddn✯)`, on peut calculer *age* en fonction de *ddn* par le calcul : *age = ddn - today()*.

Méthode

Chaque fois que c'est possible, on remplacera un attribut par une méthode si cela permet de supprimer une DF sans perdre d'information.

Cette solution est à privilégier *a priori* sur une décomposition.

En relationnel, on supprimera l'attribut de la table et on ajoutera une vue permettant de le retrouver.

Exemple : Exemple simple

Soit la relation en 2NF :

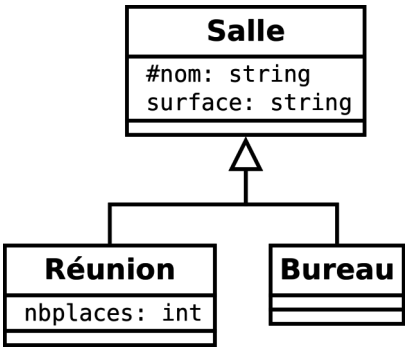
```
1 personne (#numsecu, nom, prenom, ddn, age) avec ddn → age
```

On remplacera cette relation par une relation en 3NF et une vue :

```
1 Personne (#numsecu, nom, prenom, ddn)
2 vPersonne (#numsecu, nom, prenom, ddn, age) avec  age = ddn - today()
```

Exemple : Un exemple plus approfondi

Soit le schéma UML suivant :



Une transformation de l'héritage par la classe mère nous donnerait :

```
1  Salle (#nom:string, surface:string, nbplaces:int, type:{Réunion|Bureau})
2  DF : nom → surface, type, nbplaces et nbplaces → type
```

On peut donc supprimer l'attribut type et aboutir à la relation en 3NF :

```
1  Salle (#nom:string, surface:string, nbplaces:int)
2  vSalle (nom, surface, nbplaces, type) avec  si nbplace=Null  alors type=Bureau
     sinon type=Réunion
```

e) Synthèse : Processus de conception d'une base de données normalisée

1. Analyse et clarification du problème posé
2. Modélisation conceptuelle en UML
   Résultat : MCD_1
3. Traduction en relationnel en utilisant les règles de passage UML vers relationnel
   Résultat : MLD_1
4. Établissement des DF sous la forme de la fermeture transitive pour chaque relation du modèle
   Résultat : MLD_1 avec F+
5. Établissement de la forme normale
   Résultat : MLD_1 avec F+ et NF
6. Décomposition des relations jusqu'à arriver en 3NF en préservant les DF
   Résultat : MLD_2 avec F+ et 3NF
7. Rétro-conception du modèle UML correspondant
   Résultat : MCD_2
8. Implémentation en SQL du modèle MLD_2

* *

*

La normalisation permet de décomposer un schéma relationnel afin d'obtenir des relations non redondantes.

La 3NF✯ est souhaitable car toujours possible à obtenir, sans perte d'information et sans perte de DF✯. La BCNF✯ est également indiquée, car elle est un peu plus puissante, et plutôt plus simple que la 3NF.

La BCNF✯ n'est pas encore suffisante pour éliminer toutes les redondances. Il existe pour cela les 4NF✯ et 5NF✯ qui ne sont pas abordées dans ce cours. Notons également que les cas de non-4NF et de non-5NF sont assez rares dans la réalité.

2. Exemple de synthèse : MCD-Relationnel-Normalisation-SQL

Problème posé

Soit un modèle conceptuel représentant :

- un type d'entité "chercheur", identifié par le numéro de sécurité sociale, et possédant les autres propriétés suivantes : le nom, le nom de l'université à laquelle il appartient, la ville dans laquelle est basée cette université.
- un type d'entité "professeur", héritant de "chercheur"
- un type d'entité "doctorant", héritant de "chercheur"
- une association de type "encadrement" entre professeur et doctorant (un professeur pouvant encadrer plusieurs doctorants et un doctorant n'ayant qu'un et un seul directeur de thèse).

Afin de réaliser le modèle de données :

1. Dessiner le modèle conceptuel
2. Traduire le modèle conceptuel en modèle logique relationnel.
3. Après avoir identifié les DF, normaliser le modèle relationnel en BCNF.
4. Ecrire les instructions SQL de création d'un tel modèle.

a) Première étape : Modélisation conceptuelle

Méthode : Modélisation conceptuelle : Entité Chercheur

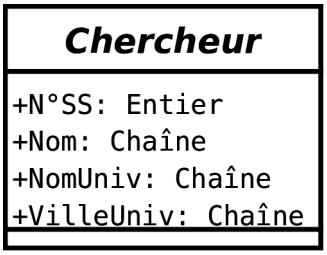


Image 31 Conception UML (1/4)

Méthode : Modélisation conceptuelle : Entité Professeur

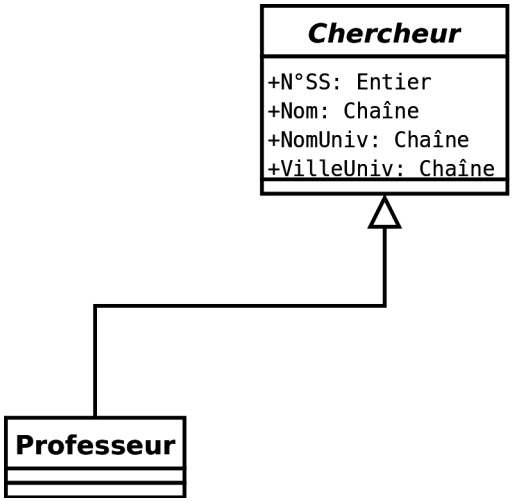


Image 32 Conception UML (2/4)

Méthode : Modélisation conceptuelle : Entité Doctorant

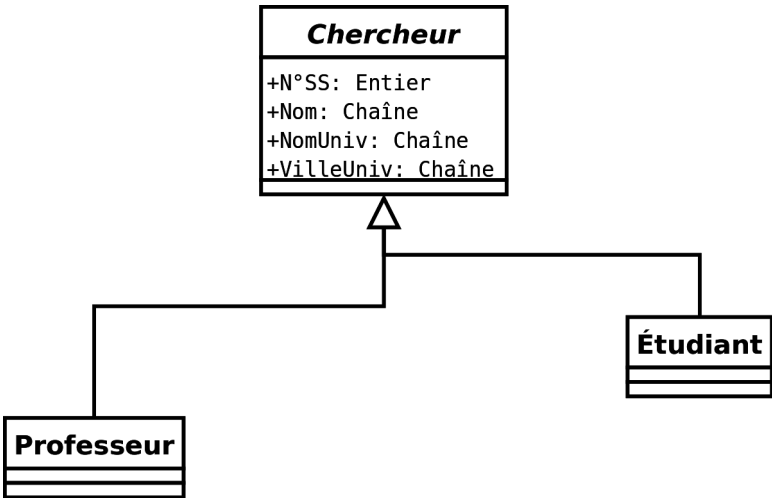


Image 33 Conception UML (3/4)

Méthode : Modélisation conceptuelle : Association Encadrement

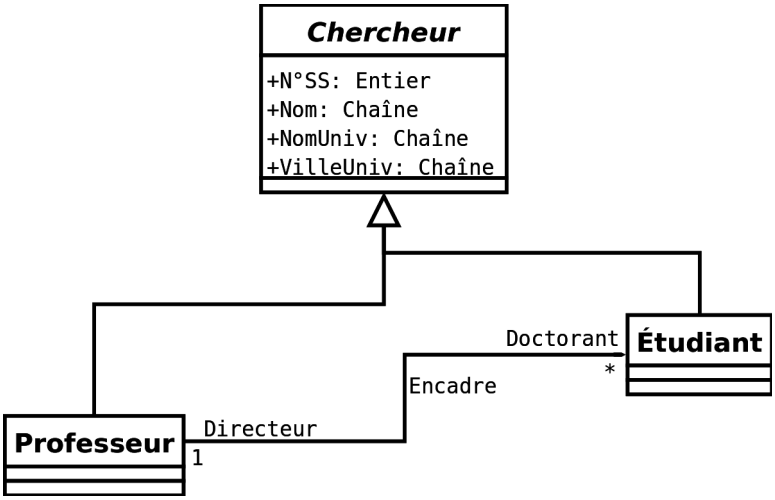


Image 34 Conception UML (4/4)

b) Deuxième étape : Modèle relationnel

Méthode : Modèle relationnel : Héritage

Choix de transformation de l'héritage : L'héritage est exclusif (les professeurs ne sont plus doctorants), mais pas complet, **car l'association Encadre n'est pas symétrique**. On choisit donc un héritage par les classes filles (Chercheur étant par ailleurs abstrait).

Méthode : Modèle relationnel : Entité Professeur

```
1  Professeur (#N°SS:int(13), Nom:char(20), NomUniv:char(50), VilleUniv:char(20))
```

Méthode : Modèle relationnel : Entité Doctorant

```
1  Professeur (#N°SS:int(13), Nom:char(20), NomUniv:char(50), VilleUniv:char(20))
2  Doctorant (#N°SS:int(13), Nom:char(20), NomUniv:char(50), VilleUniv:char(20))
```

Méthode : Modèle relationnel : Association EncadrePar

```
1 Professeur (#N°SS:int(13), Nom:char(20), NomUniv:char(50), VilleUniv:char(20))
2 Doctorant (#N°SS:int(13), Nom:char(20), NomUniv:char(50), VilleUniv:char(20),
  EncadrePar=>Professeur)
```

c) Troisième étape : Dépendances fonctionnelles

Méthode : Dépendances fonctionnelles : DF évidente (professeur)

- Professeur.N°SS → Nom, NomUniv, VilleUniv

Méthode : Dépendances fonctionnelles : DF évidente (doctorant)

- Professeur.N°SS → Nom, NomUniv, VilleUniv
- Doctorant.N°SS → Nom, NomUniv, VilleUniv, EncadrePar

Méthode : Dépendances fonctionnelles : DF déduites du sens des propriétés

Connaissant l'université, on connaît la ville, donc :

- Professeur.N°SS → Nom, NomUniv, VilleUniv
- Professeur.NomUniv → VilleUniv
- Doctorant.N°SS → Nom, NomUniv, VilleUniv, EncadrePar
- Doctorant.NomUniv → VilleUniv

d) Quatrième étape : Formes normales

Méthode : Forme normale : 1NF

Le schéma est en 1NF (clés et attributs atomique).

Méthode : Forme normale : 2NF

Le schéma est en 2NF (la clé est composée d'un seul attribut)

Méthode : Forme normale : 3NF

La schéma n'est pas en 3NF : NomUniv → VilleUniv

e) Cinquième étape : Décomposition

Méthode : Normalisation en BCNF : Résultat

```
1 Professeur (#N°SS:int(13), Nom:char(20), NomUniv=>Univ)
2 Doctorant (#N°SS:int(13), Nom:char(20), NomUniv=>Univ, EncadrePar=>Professeur)
3 Univ (#NomUniv:char(50), VilleUniv:char(20))
```

Méthode : Normalisation en BCNF : Vérification

Le modèle est bien en BCNF, toutes les DF ont pour source une clé.

Méthode : Normalisation en BCNF : Conservation des DF

La transformation préserve les DF car :

- N°SS → NomUniv et Univ.Nom → Ville
- Donc N°SS → Univ.Ville (par transitivité)

f) Sixième étape : Implémentation SQL LDD

Méthode : Implémentation SQL : Professeur

```
1  Create Table Professeur (
2   N°SS INTEGER(13) PRIMARY KEY,
3  Nom CHAR(20) NOT NULL,
4  NomUniv CHAR(50) REFERENCES Univ(Nom));
```

Méthode : Implémentation SQL : Doctorant

```
1  Create Table Professeur (
2   N°SS INTEGER(13) PRIMARY KEY,
3  Nom CHAR(20) NOT NULL,
4  NomUniv CHAR(50) REFERENCES Univ(Nom));
```

```
1  Create Table  Doctorant (
2   N°SS INTEGER(13) PRIMARY KEY,
3  Nom CHAR(20) NOT NULL,
4  NomUniv CHAR(50) REFERENCES Univ(Nom),
5  EncadrePar INTEGER(13) REFERENCES Professeur(N°SS));
```

Méthode : Implémentation SQL : Univ

```
1  Create Table Professeur (
2   N°SS INTEGER(13) PRIMARY KEY,
3  Nom CHAR(20) NOT NULL,
4  NomUniv CHAR(50) REFERENCES Univ(Nom));
```

```
1  Create Table  Doctorant (
2   N°SS INTEGER(13) PRIMARY KEY,
3  Nom CHAR(20) NOT NULL,
4  NomUniv CHAR(50) REFERENCES Univ(Nom),
5  EncadrePar INTEGER(13) REFERENCES Professeur(N°SS));
```

```
1  Create Table Univ (
2  Nom CHAR(50) PRIMARY KEY,
3   Ville CHAR(20) );
```

B. Exercices

1. Project manager

[15 min]

Soit la table suivante (Emp signifie *Employee*, Pro signifie *Project* et Man signifie *Manager*).

Emp	EmpName	Pro	ProName	Man	ManName
E01	John	P1	Eco	M1	Becky
E02	Mary	P2	Admin	M2	Joe
E03	Mark	P2	Admin	M2	Joe
E04	Travis	P3	Educ	M1	Becky

Soit les dépendances fonctionnelles suivantes :

- Emp → EmpName
- Emp → Pro
- Pro → ProName
- Pro → Man
- Pro → ManName
- Man → ManName

Question 1

Montrer que ce modèle n'est pas en 3NF.

Question 2

Proposer un modèle équivalent en 3NF.

2. Objectifs II

[15 min]

Vous êtes le président de l'association "Objectifs", dont l'objet est d'aider ses étudiants membres à mener des projets dans le cadre de leurs études. Pour le moment cette association gère tous ces projets avec un logiciel de type tableur.

Numéro	Projet / Tâche	Spécialités	Chef de projet	Participant tâche	Dates	Partenaire	
1	La nuit du Picolo	Logistique	Paul Densmore ; Spécialiste Musique		25 décembre	1666 (Brasseur)	Bières offertes
1,1	Gestion			Barbara Krieger ; Spécialiste Sport			
2	Escalade de l'Everest	Voyages	Paul Manzarek		Intersemestre	Pentathlon 1666 (Brasseur)	Fourniture de matériel Publicité
3	Tournoi de Volley-Ball	Sport	Paul Manzarek		15/5 - 23/5		
3.1	Recrutement des équipes			Barbara Krieger ; Spécialiste Sport			
3.2	Gestion			Paul Densmore ; Spécialiste Musique			

Exemple de fichier de gestion des projets de l'association Objectifs

La concision et la clarté de la rédaction, ainsi que la bonne mobilisation des concepts et de la terminologie du domaine des bases de données seront intégrées à l'évaluation.

Question 1

On considérera ce tableau comme une relation, et on admettra que cette extension est suffisante pour l'interprétation des données :

1. **Démontrer** que cette relation ne peut admettre qu'une seule clé.
2. **Démontrer** que la 1NF★ n'est pas respectée à plusieurs reprises (on peut trouver jusque six causes empêchant la 1NF, proposez-en un maximum).

Question 2

On propose une première décomposition pour que le modèle soit en 1NF.

```
1  Projet (#Num, Nom, Debut, Fin, Specialite, CDP-Prenom, CDP-Nom, CDP-Specialite)
2  Tache (#NumProjet=>Projet, #NumTache, Nom, Participant-Prenom, Participant-Nom,
       Participant-Specialite)
3  Partenaire (#Nom, Description)
4  Partenariat (#Partenaire=>Partenaire, #Projet=>Projet, role)
```

1. Montrer que ce modèle n'est pas en 3NF et lister l'ensemble des DFE responsables.
2. Proposer une décomposition en 3NF.

3. Jeu de construction

[30 min]

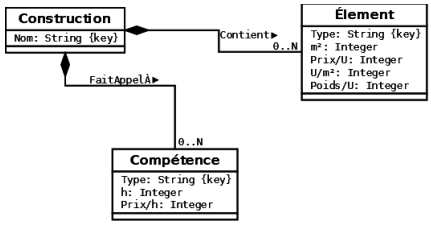
Soit le diagramme UML ci-après décrivant des **constructions** composées d'**éléments** et faisant appel à des **compétences** pour leur montage.

Par exemple une "Maison de type I" contiendra :

- 100 m² de "Brique traditionnelle" (Type de l'élément), pour un prix de 2 euros par unité (Prix/U), 12 briques par m² (U/m²) et 100g / brique (Poids/U)
- 125 m² de "Tuile plate" (Type de l'élément), pour un prix de 4 euros par unité (Prix/U), 24 tuile par m² (U/m²) et 75g / tuile (Poids/U)

et fera appel à :

- 500h de travail de "Maçon" (Type de compétence), pour un tarif horaire de 20€/h
- 425h de travail de "Couvreur" (Type de compétence), pour un tarif horaire de 30€/h
- 75h de "Chef de chantier" (Type de compétence), pour un tarif horaire de 45€/h



Modèle UML

Question 1

Effectuer le passage au relationnel de ce modèle.

Question 2

Sachant que deux mêmes types d'élément ont **toujours** les mêmes caractéristiques (Prix/U, U/m², Poids/U) et que deux mêmes types de compétence ont **toujours** les mêmes prix (Prix/h), établir la fermeture transitive des DFE et la forme normale de ce modèle (justifier).

Donner un exemple de données redondantes.

Indice :

> *Deux types de maison ont évidemment des quantités d'éléments (m²) et de compétences (h) différentes.*

Question 3

Normaliser le modèle en 3NF (justifier, et montrer que la transformation est sans perte).

Question 4

En repartant de la forme normalisée en 3NF, rétro-concevoir un modèle UML qui aurait permis d'aboutir directement à ce schéma non redondant.

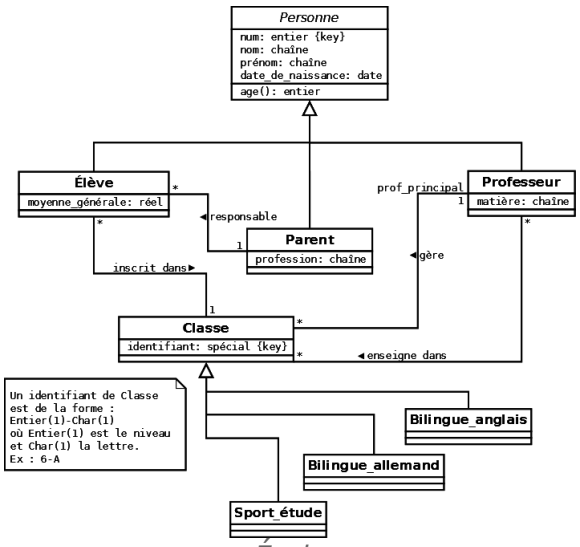
4. À l'école

[15 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).



École

Question 1

À partir du schéma UML proposé, réaliser le passage au modèle relationnel en justifiant les choix pour les deux héritages.

Question 2

Montrer que ce schéma n'est pas en première forme normale, et proposer une correction du schéma relationnel.

Gestion des transactions pour la fiabilité et la concurrence

IX

A. Cours

Les transactions sont une réponse générale aux problèmes de fiabilité et d'accès concurrents dans les BD, et en particulier dans les BD en mode client-serveur. Elles sont le fondement de toute implémentation robuste d'une BD. Un SGBDR ne fonctionne nativement qu'en mode transactionnel.

1. Principes des transactions

Objectifs

Comprendre les principes et l'intérêt des transactions

a) Problématique des pannes et de la concurrence

Une BD✯ est un ensemble persistant de données organisées qui a en charge la préservation de la cohérence de ces données. Les données sont cohérentes si elles respectent l'ensemble des contraintes d'intégrité spécifiées pour ces données : contraintes de domaine, intégrité référentielle, dépendances fonctionnelles...

La cohérence des données peut être remise en cause par deux aspects de la vie d'une BD :

- **La défaillance**

  Lorsque le système tombe en panne alors qu'un traitement est en cours, il y a un risque qu'une partie seulement des instructions prévues soit exécutée, ce qui peut conduire à des incohérences. Par exemple pendant une mise à jour en cascade de clés étrangères suite au changement d'une clé primaire.

- **La concurrence**

  Lorsque deux accès concurrents se font sur les données, il y a un risque que l'un des deux accès rende l'autre incohérent. Par exemple si deux utilisateurs en réseau

modifient une donnée au même moment, seule une des deux mises à jour sera effectuée.

La gestion de transactions par un SGBD✯ est à la base des mécanismes qui permettent d'assurer le maintien de la cohérence des BD. C'est à dire encore qu'il assure que toutes les contraintes de la BD seront toujours respectées, même en cas de panne et même au cours d'accès concurrents.

b) Notion de transaction

Définition : Transaction

Une transaction est une unité logique de travail, c'est à dire une séquence d'instructions, dont l'exécution assure le passage de la BD✯ d'un état cohérent à un autre état cohérent.

Fondamental: Cohérence des exécutions incorrectes

La transaction assure le maintien de la cohérence des données que son exécution soit correcte ou incorrecte.

Exemple : Exemples d'exécutions incorrectes

L'exécution d'une transaction peut être incorrecte parce que :

- Une panne a lieu
- Un accès concurrent pose un problème
- Le programme qui l'exécute en a décidé ainsi

c) Déroulement d'une transaction

1. DEBUT
2. TRAITEMENT
   - Accès aux données en lecture
   - Accès aux données en écriture
3. FIN
   - Correcte : Validation des modifications
   - Incorrecte : Annulation des modifications

Fondamental

Tant qu'une transaction n'a pas été terminée correctement, elle doit être assimilée à une tentative ou une mise à jour virtuelle, elle reste incertaine. Une fois terminée correctement la transaction ne peut plus être annulée par aucun moyen.

d) Propriétés ACID d'une transaction

Une transaction doit respecter quatre propriétés fondamentales :

- **L'atomicité**

  Les transactions constituent l'unité logique de travail, toute la transaction est exécutée ou bien rien du tout, mais jamais une partie seulement de la transaction.

- **La cohérence**

  Les transactions préservent la cohérence de la BD✯, c'est à dire qu'elle transforme la BD d'un état cohérent à un autre (sans nécessairement que les états intermédiaires internes de la BD au cours de l'exécution de la transaction respectent cette cohérence)

- **L'isolation**

  Les transactions sont isolées les unes des autres, c'est à dire que leur exécution est indépendante des autres transactions en cours. Elles accèdent donc à la BD comme si elles étaient seules à s'exécuter, avec comme corollaire que les résultats intermédiaires d'une transaction ne sont jamais accessibles aux autres transactions.

- **La durabilité**

Les transactions assurent que les modifications qu'elles induisent perdurent, même en cas de défaillance du système.

Remarque

Les initiales de Atomicité, Cohérence, Isolation et Durabilité forme le mot mnémotechnique ACID.

e) Journal des transactions

Définition : Journal

Le journal est un fichier système qui constitue un espace de stockage redondant avec la BD★. Il répertorie l'ensemble des mises à jour faites sur la BD (en particulier les valeurs des enregistrements avant et après mise à jour). Le journal est donc un historique **persistant** (donc en mémoire secondaire) qui mémorise tout ce qui se passe sur la BD.

Le journal est indispensable pour la validation (COMMIT), l'annulation (ROLLBACK) et la reprise après panne de transactions.

Synonymes : Log

2. Manipulation de transactions en SQL

Objectifs

Connaître les syntaxes SQL standard, PostgreSQL, Oracle et Access pour utiliser des transactions

a) Transactions en SQL

Introduction

Le langage SQL★ fournit trois instructions pour gérer les transactions.

Syntaxe : Début d'une transaction

```
1 BEGIN TRANSACTION (ou BEGIN) ;
```

Cette syntaxe est optionnelle (voire inconnue de certains SGBD★), une transaction étant débutée de façon **implicite** dès qu'instruction est initiée sur la BD★.

Syntaxe : Fin correcte d'une transaction

```
1 COMMIT TRANSACTION (ou COMMIT) ;
```

Cette instruction SQL signale la fin d'une transaction couronnée de succès. Elle indique donc au gestionnaire de transaction que l'unité logique de travail s'est terminée dans un état cohérent est que les données peuvent effectivement être modifiées de façon durable.

Syntaxe : Fin incorrecte d'une transaction

```
1 ROLLBACK TRANSACTION (ou ROLLBACK) ;
```

Cette instruction SQL signale la fin d'une transaction pour laquelle quelque chose s'est mal passé. Elle indique donc au gestionnaire de transaction que l'unité logique de travail s'est terminée dans un état potentiellement incohérent et donc que les données ne doivent pas être modifiées en annulant les modifications réalisées au cours de la transaction.

Remarque : Programme

Un programme est généralement une séquence de plusieurs transactions.

b) Mini-TP : Transaction en SQL standard sous PostgreSQL

1. Se connecter à une base de données : `psql mydb`
2. Créer une table `test` : `CREATE TABLE test (a integer);`

Question 1

1. Commencer une transaction : `BEGIN TRANSACTION;`
2. Insérer les deux valeurs 1 et 2 dans la table : `INSERT INTO...`
3. Valider la transaction : `COMMIT`
4. Vérifier que les valeurs sont bien dans la table : `SELECT * FROM ...`

Question 2

1. Commencer une transaction : `BEGIN TRANSACTION;`
2. Insérer les deux valeurs 3 et 4 dans la table : `INSERT INTO...`
3. Annuler la transaction : `ROLLBACK`
4. Vérifier que les valeurs ne sont pas dans la table : `SELECT * FROM ...`

c) Exemple : Transaction sous Oracle en SQL

Exemple : Exemple en SQL sous Oracle

```
1  INSERT INTO test (a) VALUES (1);
2  COMMIT;
3
```

Attention : BEGIN implicite sous Oracle

La commande `BEGIN;` ou `BEGIN TRANSACTION;` ne peut pas être utilisé sous Oracle (la commande `BEGIN` est réservée à l'ouverture d'un bloc PL/SQL).

Toute commande SQL LMD (`INSERT`, `UPDATE` ou `DELETE`) démarre par défaut une transaction, la commande `BEGIN TRANSACTION` est donc implicite.

d) Exemple : Transaction sous Oracle en PL/SQL

Exemple : Exemple en PL/SQL sous Oracle

```
1  BEGIN
2    INSERT INTO test (a) VALUES (1);
3    COMMIT;
4  END;
5
```

e) Exemple : Transaction sous Access en VBA

Exemple : Exemple de transfert entre deux comptes en VBA sous Access

```
1  Sub MyTransaction
2    BeginTrans
3      CurrentDb.CreateQueryDef("", "INSERT INTO test (a) VALUES (1)").Execute
4    CommitTrans
5  End Sub
```

Remarque : VBA et transactions

Sous Access, il n'est possible de définir des transactions sur plusieurs objets requêtes qu'en VBA.

f) Mode Autocommit

Attention : Autocommit

La plupart des clients et langages d'accès aux bases de données proposent un mode *autocommit* permettant d'encapsuler chaque instruction dans une transaction. Ce mode revient à avoir un `COMMIT` implicite après chaque instruction.

Ce mode doit être désactivé pour permettre des transactions portant sur plusieurs instructions.

Oracle

- Sous Oracle SQL*Plus : `SET AUTOCOMMIT ON` et `SET AUTOCOMMIT OFF`
- Sous Oracle SQL Developper : Menu `Outils > Préférences > Base de données > Paramètres de feuille de calcul > Validation automatique dans une feuille de calcul`

PostgreSQL

Avec psql :

- `\set AUTOCOMMIT on`
- `\set AUTOCOMMIT off`

Si l'*autocommit* est activé, il est néanmoins possible de démarrer une transaction sur plusieurs lignes en exécutant un `BEGIN TRANSACTION` explicite : `BEGIN ; ... ; COMMIT ;`.

Ainsi deux modes sont possibles :

- *Autocommit* activé : BEGIN explicites, COMMIT implicites
- *Autocommit* désactivé : BEGIN implicites, COMMIT explicites

Access

Sous Access, toute requête portant sur plusieurs lignes d'une table est encapsulée dans une transaction.

Ainsi par exemple la requête `UPDATE Compte SET Solde=Solde*6,55957` est exécutée dans une transaction et donc, soit toutes les lignes de la table Compte seront mises à jour, soit aucune (par exemple en cas de panne pendant .

g) Exercice

Quel est le résultat de l'exécution suivante sous Oracle ?

```
1  SET AUTOCOMMIT OFF
2  CREATE TABLE T1 (A INTEGER);
3
4  INSERT INTO T1 VALUES (1);
5  INSERT INTO T1 VALUES (2);
6  INSERT INTO T1 VALUES (3);
7  COMMIT;
8  INSERT INTO T1 (4);
9  INSERT INTO T1 (5);
10 ROLLBACK;
11 SELECT SUM(A) FROM T1
```

h) Exercice

Combien de lignes renvoie l'avant-dernière instruction SELECT ?

```
1  SET AUTOCOMMIT OFF
2  CREATE TABLE T1 (A INTEGER);
3
4  INSERT INTO T1 VALUES (1);
5  INSERT INTO T1 VALUES (1);
6  INSERT INTO T1 VALUES (1);
7  SELECT * FROM T1;
8  COMMIT;
```

i) Exercice

Combien de tables sont créées par cette série d'instruction ?

```
1  SET AUTOCOMMIT OFF
2  CREATE TABLE T1 (A INTEGER);
3  CREATE TABLE T2 (A INTEGER);
4  CREATE TABLE T3 (A INTEGER);
5  INSERT INTO T1 VALUES (1);
6  INSERT INTO T2 VALUES (1);
7  INSERT INTO T3 VALUES (1);
8  ROLLBACK;
```

3. Fiabilité et transactions

Objectifs

Appréhender la gestion des pannes dans les SGBD.
Comprendre la réponse apportée par la gestion des transactions.

a) Les pannes

Une BD✯ est parfois soumise à des défaillances qui entraînent une perturbation, voire un arrêt, de son fonctionnement.

On peut distinguer deux types de défaillances :

- **Les défaillances système**

  ou défaillances douces (*soft crash*), par exemple une coupure de courant ou une panne réseau. Ces défaillances affectent toutes les transactions en cours de traitement, mais pas la BD au sens de son espace de stockage physique.

- **Les défaillances des supports**

  ou défaillances dures (*hard crash*), typiquement le disque dur sur lequel est stockée la BD. Ces défaillances affectent également les transactions en cours (par rupture des accès aux enregistrements de la BD), mais également les données elles-mêmes.

Remarque : Annulation des transactions non terminées

Lorsque le système redémarre après une défaillance, toutes les transactions qui étaient en cours d'exécution (pas de COMMIT) au moment de la panne sont annulés (ROLLBACK imposé par le système).

Cette annulation assure le retour à un état cohérent, en vertu des propriétés ACID✯ des transactions.

Remarque : Ré-exécution des transactions terminées avec succès

Au moment de la panne certaines transactions étaient peut-être terminées avec succès (COMMIT) mais non encore (ou seulement partiellement) enregistrées dans la BD (en mémoire volatile, tampon, etc.). Lorsque le système redémarre il doit commencer par rejouer ces transactions, qui assurent un état cohérent de la BD plus avancé.

Cette reprise des transactions après COMMIT est indispensable dans la mesure où c'est bien l'instruction COMMIT qui assure la fin de la transaction et donc la **durabilité**. Sans cette gestion, toute transaction pourrait être remise en cause.

Remarque : Unité de reprise

Les transactions sont des unités de travail, et donc également de reprise.

Remarque : Défaillance des supports

Tandis que la gestion de transactions et de journal permet de gérer les défaillances systèmes, les défaillances des supports ne pourront pas toujours être gérés par ces seuls mécanismes.

Il faudra leur adjoindre des procédures de **sauvegarde** et de restauration de la BD pour être capable au pire de revenir dans un état antérieur cohérent et au mieux de réparer complètement la BD (cas de la réplication en temps réel par exemple).

b) Point de contrôle

Définition : Point de contrôle

Un point de contrôle est une écriture dans le journal positionnée automatiquement par le système qui établit la liste de toutes les transactions en cours (au moment où le point de contrôle est posé) et force la sauvegarde des données alors en mémoire centrale dans la mémoire secondaire.

Le point de contrôle est positionné à intervalles de temps ou de nombre d'entrées.

Le dernier point de contrôle est le point de départ d'une reprise après panne, dans la mesure où c'est le dernier instant où toutes les données ont été sauvegardées en mémoire non volatile.

Synonymes : Syncpoint

c) Ecriture en avant du journal

Fondamental

On remarquera que pour que la reprise de panne soit en mesure de rejouer les transactions, la première action que doit effectuer le système au moment du COMMIT est l'écriture dans le journal de cette fin correcte de transaction. En effet ainsi la transaction pourra être rejouée, même si elle n'a pas eu le temps de mettre effectivement à jour les données dans la BD, puisque le journal est en mémoire secondaire.

d) Reprise après panne

Introduction

Le mécanisme de reprise après panne s'appuie sur le journal et en particulier sur l'état des transactions au moment de la panne et sur le dernier point de contrôle.

Le schéma ci-après illustre les cinq cas de figure possibles pour une transaction au moment de la panne.

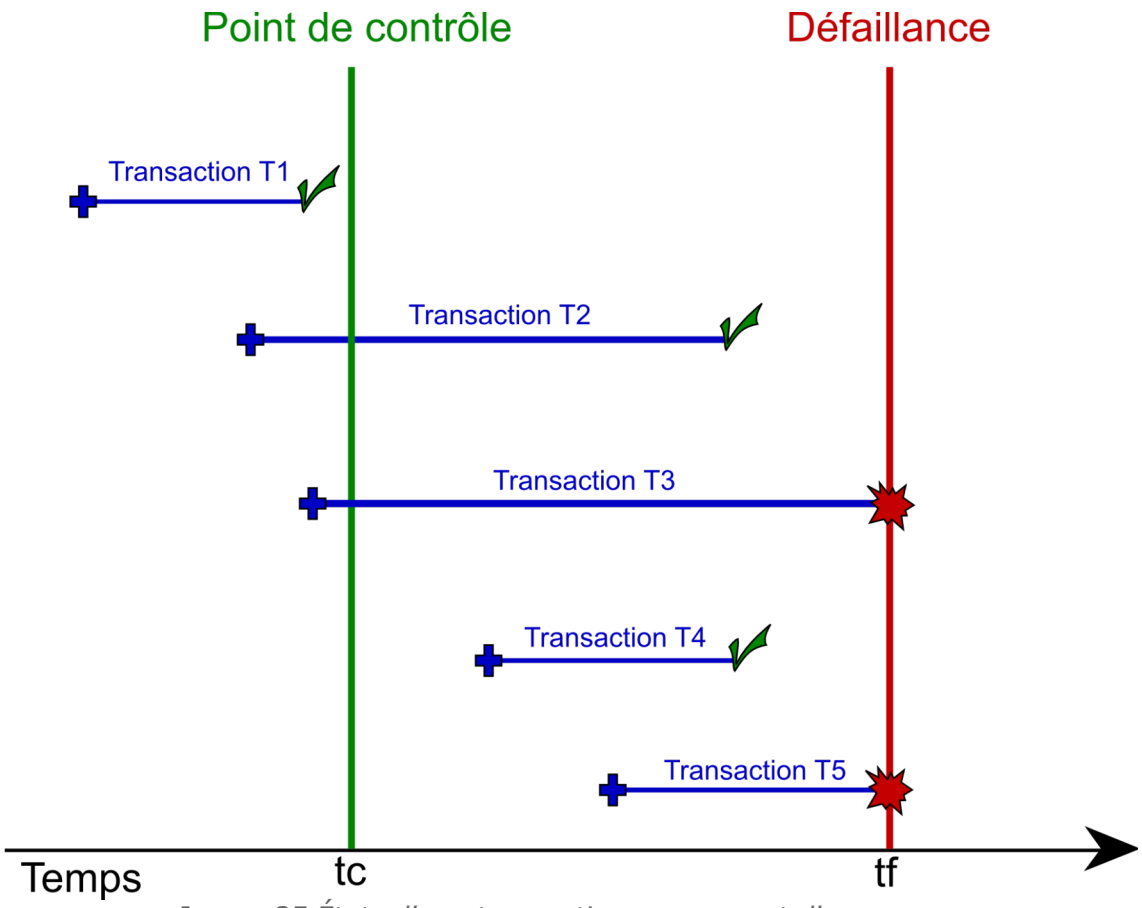


Image 35 États d'une transaction au moment d'une panne

- **Transactions de type T1**

  Elles ont débuté et se sont terminées avant tc. Elles n'interviennent pas dans le processus de reprise.
- **Transactions de type T2**

  Elles ont débuté avant tc et se sont terminées entre tc et tf. Elles devront être rejouées (il n'est pas sûr que les données qu'elles manipulaient aient été correctement inscrites en mémoire centrale, puisque après tc, or le COMMIT impose la durabilité).
- **Transactions de type T3**

  Elles ont débuté avant tc, mais n'était pas terminées à tf. Elles devront être annulées (pas de COMMIT).
- **Transactions de type T4**

  Elles ont débuté après tc et se sont terminées avant tf. Elles devront être rejouées.
- **Transactions de type T5**

  Elles ont débuté après tc et ne se sont pas terminées. Elles devront être annulées.

Remarque

Les transactions sont des unités d'intégrité.

e) Mini-TP : Simulation d'une panne sous PostgreSQL

1. Se connecter à une base de données : `psql mydb`
2. Créer une table `test` : `CREATE TABLE test (a integer);`

Question

1. Commencer une transaction : `BEGIN TRANSACTION;`
2. Insérer les deux valeurs 1 et 2 dans la table : `INSERT INTO...`
3. Vérifier que les valeurs sont bien dans la table : `SELECT * FROM ...`
4. Simuler un crash en fermant le terminal : `ROLLBACK` du système
5. Se reconnecter et vérifier que les valeurs **ne sont plus** dans la table : `SELECT * FROM ...`

f) Exemple : Transfert protégé entre deux comptes

Exemple : Transfert entre deux comptes en SQL standard sous PostgreSQL

```
1  BEGIN;
2  UPDATE Compte1 SET Solde=Solde+100 WHERE Num=1;
3  UPDATE Compte2 SET Solde=Solde-100 WHERE Num=1;
4  COMMIT;
5  /
```

Exemple : Transfert entre deux comptes en PL/SQL sous Oracle

```
1  CREATE OR REPLACE PROCEDURE myTransfC1C2
2  IS
3  BEGIN
4    UPDATE Compte1 SET Solde=Solde+100 WHERE Num=1;
5    UPDATE Compte2 SET Solde=Solde-100 WHERE Num=1;
6   COMMIT;
7  END;
8  /
```

Exemple : Transfert entre deux comptes en VBA sous Access

```
1  Sub myTransfC1C2
2    BeginTrans
3      CurrentDb.CreateQueryDef("", "UPDATE Compte1 SET Solde=Solde+100 WHERE
   Num=1").Execute
4      CurrentDb.CreateQueryDef("", "UPDATE Compte2 SET Solde=Solde-100 WHERE
   Num=1").Execute
5    CommitTrans
6  End Sub
```

Fondamental: Transfert protégé

Pour les trois exemples ci-avant le transfert est protégé au sens où soit les deux `UPDATE` seront exécutés , soit aucun.

En cas de panne pendant la transaction, le transfert sera annulé (`ROLLBACK` système), mais en aucun cas un des deux comptes ne pourra être modifié sans que l'autre le soit (ce qui aurait entraîné une perte ou un gain sur la somme des deux comptes).

g) Exercice

Pour faire un transfert sécurisé d'un point de vue transactionnel de 100€ du compte bancaire C1 vers le compte bancaire C2 pour le compte numéro 12, quelle est la série d'instructions correcte (en mode `autocommit off`)?

○	`UPDATE C1 SET Solde=Solde-100 WHERE Numero=12;`
	`ROLLBACK;`
	`UPDATE C2 SET Solde=Solde+100 WHERE Numero=12;`
	`COMMIT;`

○	`UPDATE C1 SET Solde=Solde-100 WHERE Numero=12`
	`UPDATE C2 SET Solde=Solde+100 WHERE Numero=12`
	`ROLLBACK;`

○	`UPDATE C1 SET Solde=Solde-100 WHERE Numero=12;`
	`COMMIT;`
	`UPDATE C2 SET Solde=Solde+100 WHERE Numero=12;`
	`COMMIT;`

○	`UPDATE C1 SET Solde=Solde-100 WHERE Numero=12;`
	`UPDATE C2 SET Solde=Solde+100 WHERE Numero=12;`
	`COMMIT;`

○	`UPDATE C1 SET Solde=Solde-100 WHERE Numero=12;`
	`ROLLBACK;`
	`UPDATE C2 SET Solde=Solde-100 WHERE Numero=12;`
	`ROLLBACK;`

h) Complément : Algorithme de reprise UNDO-REDO

Introduction

L'algorithme suivant permet d'assurer une reprise après panne qui annule et rejoue les transactions adéquates.

```
1  1. SOIT deux listes REDO et UNDO
2    1a. Initialiser la liste REDO à vide
3    1b. Initialiser la liste UNDO avec toutes les transactions en cours au dernier
   point de contrôle
4
5  2. FAIRE une recherche en avant dans le journal, à partir du point de contrôle
6    2a. SI une transaction T est commencée ALORS  ajouter T à UNDO
7    2b. SI une transaction T est terminée avec succès alors déplacer T de UNDO à
   REDO
8
9  3. QUAND la fin du journal est atteinte
10   3a. Annuler les transactions de la liste UNDO (reprise en arrière)
11   3b. Rejouer les transactions de la liste REDO (reprise en avant)
12
13 4. TERMINER la reprise et redevenir disponible pour de nouvelles instructions
```

Exemple

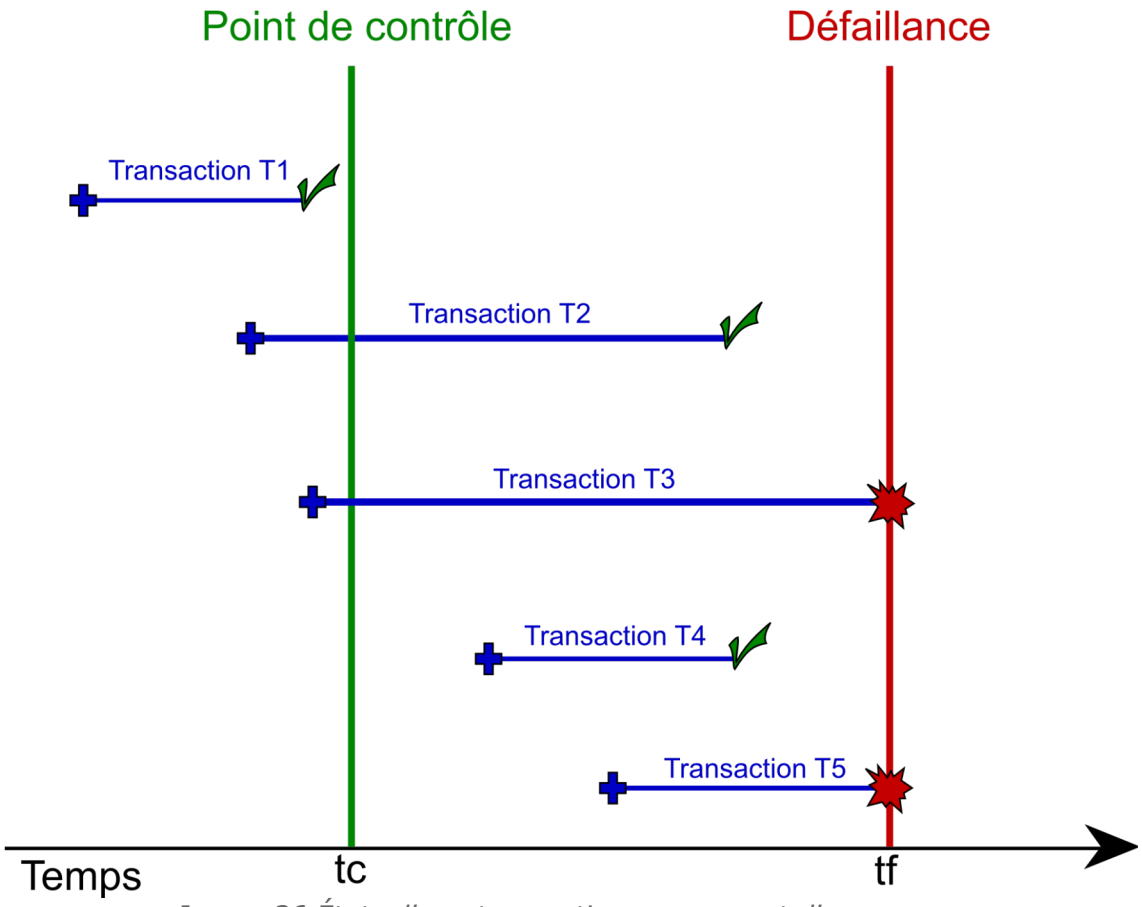


Image 36 États d'une transaction au moment d'une panne

- **Transactions de type T1**

  Non prises en compte par l'algorithme.
- **Transactions de type T2**

  Ajoutées à la liste UNDO (étape 1b) puis déplacée vers REDO (étape 2b) puis rejouée (étape 3b).
- **Transactions de type T3**

  Ajoutées à la liste UNDO (étape 1b) puis annulée (étape 3a).
- **Transactions de type T4**

  Ajoutées à la liste UNDO (étape 2a) puis déplacée vers REDO (étape 2b) puis rejouée (étape 3b).
- **Transactions de type T5**

  Ajoutées à la liste UNDO (étape 2a) puis annulée (étape 3a).

* *

*

On voit que la gestion transactionnelle est un appui important à la reprise sur panne, en ce qu'elle assure des états cohérents qui peuvent être restaurés.

4. Concurrence et transactions

Objectifs

Appréhender la gestion de la concurrence dans les SGBD.
Comprendre la réponse apportée par la gestion des transactions.

a) Trois problèmes soulevés par la concurrence.

Introduction

Nous proposons ci-dessous trois problèmes posés par les accès concurrents des transactions aux données.

i Perte de mise à jour

Temps	Transaction A	Transaction B
t1	LIRE T	
t2	...	LIRE T
t3	UPDATE T	...
t4	...	UPDATE T
t5	COMMIT	...
t4		COMMIT

Tableau 9 Problème de la perte de mise à jour du tuple T par la transaction A

Les transaction A et B accèdent au même tuple T ayant la même valeur respectivement à t1 et t2. Ils modifient chacun la valeur de T. Les modifications effectuées par A seront perdues puisqu'elle avait lu T avant sa modification par B.

Exemple

Temps	A : Ajouter 100	B : Ajouter 10
t1	LIRE COMPTE C=1000	
t2	...	LIRE COMPTE C=1000
t3	UPDATE COMPTE C=C+100=1100	...
t4	...	UPDATE COMPTE C=C+10=1010
t5	COMMIT C=1100	...
t6		COMMIT C=1010

Tableau 10 Doucle crédit d'un compte bancaire C

Dans cet exemple le compte bancaire vaut 1010 à la fin des deux transactions à la place de 1110.

ii Accès à des données non validées

Temps	Transaction A	Transaction B
t1		UPDATE T
t2	LIRE T	...
t3		ROLLBACK

Tableau 11 Problème de la lecture impropre du tuple T par la transaction A

La transaction A accède au tuple T qui a été modifié par la transaction B. B annule sa modification et A a donc accédé à une valeur qui n'aurait jamais dû exister (virtuelle) de T. Pire A pourrait faire une mise à jour de T **après** l'annulation par B, cette mise à jour inclurait la valeur **avant** annulation par B (et donc reviendrait à annuler l'annulation de B).

Exemple

Temps	A : Ajouter 10	B : Ajouter 100 (erreur)
t1		LIRE COMPTE C=1000
t2		UPDATE COMPTE C=C+100=1100
t3	LIRE COMPTE C=1100	...
t4	...	ROLLBACK C=1000
t5	UPDATE C C=C+10=1110	
t6	COMMIT C=1110	

Tableau 12 Annulation de crédit sur le compte bancaire C

Dans cet exemple le compte bancaire vaut 1110 à la fin des deux transactions à la place de 1010.

iii Lecture incohérente

Temps	Transaction A	Transaction B
t1	LIRE T	
t2	...	UPDATE T
t3	...	COMMIT
t4	LIRE T	

Tableau 13 Problème de la lecture non reproductible du tuple T par la transaction A

Si au cours d'une même transaction A accède deux fois à la valeur d'un tuple alors que ce dernier est, entre les deux, modifié par une autre transaction B, alors la lecture de A est inconsistente. Ceci peut entraîner des incohérences par exemple si un calcul est en train d'être fait sur des valeurs par ailleurs en train d'être mises à jour par d'autres transactions.

Remarque

Le problème se pose bien que la transaction B ait été validée, il ne s'agit donc pas du problème d'accès à des données non validées.

Exemple

Temps	A : Calcul de S=C1+C2	B : Transfert de 10 de C1 à C2
t1	LIRE COMPTE1 C1=100	
t2	...	LIRE COMPTE 1 C1=100
t3	...	LIRE COMPTE 2 C2=100
t4	...	UPDATE COMPTE 1 C1=100-10=90
t5	...	UPDATE COMPTE 2 C2=100+10=110
t6	...	COMMIT
t7	LIRE COMPTE 2 C2=110	
t8	CALCUL S S=C1+C2=210	

Tableau 14 Transfert du compte C1 au compte C2 pendant une opération de calcul C1+C2

Dans cet exemple la somme calculée vaut 210 à la fin du calcul alors qu'elle devrait valoir 200.

b) Le verrouillage

Introduction

Une solution générale à la gestion de la concurrence est une technique très simple appelée verrouillage.

Définition : Verrou

Poser un verrou sur un objet (typiquement un tuple) par une transaction signifie rendre cet objet inaccessible aux autres transactions.

Synonymes : Lock

Définition : Verrou partagé S

Un verrou partagé, noté S, est posé par une transaction lors d'un accès en **lecture** sur cet objet.

Un verrou partagé interdit aux autres transaction de poser un verrou exclusif sur cet objet et donc d'y accéder en écriture.

Synonymes : Verrou de lecture, Shared lock, Read lock

Définition : Verrou exclusif X

Un verrou exclusif, noté X, est posé par une transaction lors d'un accès en **écriture** sur cet objet.

Un verrou exclusif interdit aux autres transactions de poser tout autre verrou (partagé ou exclusif) sur cet objet et donc d'y accéder (ni en lecture, ni en écriture).

Synonymes : Verrou d'écriture, Exclusive lock, Write lock

Remarque : Verrous S multiples

Un même objet peut être verrouillé de façon partagée par plusieurs transactions en même temps. Il sera impossible de poser un verrou exclusif sur cet objet tant **qu'au moins une** transaction disposera d'un verrou S sur cet objet.

Méthode : Règles de verrouillage

Soit la transaction A voulant poser un verrou S sur un objet O

1. Si O n'est pas verrouillé alors A peut poser un verrou S
2. Si O dispose déjà d'un ou plusieurs verrous S alors A peut poser un verrou S
3. Si O dispose déjà d'un verrou X alors A ne peut pas poser de verrou S

Soit la transaction A voulant poser un verrou X sur un objet O

1. Si O n'est pas verrouillé alors A peut poser un verrou X
2. Si O dispose déjà d'un ou plusieurs verrous S ou d'un verrou X alors A ne peut pas poser de verrou X

	Verrou X présent	Verrou(s) S présent(s)	Pas de verrou présent
Verrou X demandé	Demande refusée	Demande refusée	Demande accordée
Verrou S demandé	Demande refusée	Demande accordée	Demande accordée

Tableau 15 Matrice des règles de verrouillage

Remarque : Promotion d'un verrou

Une transaction qui dispose déjà, **elle-même**, d'un verrou S sur un objet peut obtenir un verrou X sur cet objet si aucune autre transaction ne détient de verrou S sur l'objet. Le verrou est alors promu du statut partagé au statut exclusif.

c) Le déverrouillage

Définition : Déverrouillage

Lorsqu'une transaction se termine (COMMIT ou ROLLBACK) elle libère tous les verrous qu'elle a posé.

Synonymes : *Unlock*

d) Inter-blocage

Définition : Inter-blocage

L'inter-blocage est le phénomène qui apparaît quand deux transactions (ou plus, mais généralement deux) se bloquent mutuellement par des verrous posés sur les données. Ces verrous empêchent chacune des transactions de se terminer et donc de libérer les verrous qui bloquent l'autre transaction. Un processus d'attente sans fin s'enclenche alors.

Les situations d'inter-blocage sont détectées par les SGBD★ et gérées, en annulant l'une, l'autre ou les deux transactions, par un ROLLBACK système. Les méthodes utilisées sont la détection de cycle dans un graphe d'attente et la détection de délai d'attente trop long.

Synonymes : *Deadlock*, Blocage, Verrou mortel

Définition : Cycle dans un graphe d'attente

Principe de détection d'un inter-blocage par détection d'un cycle dans un graphe représentant quelles transactions sont en attente de quelles transactions (par inférence sur les verrous

posés et les verrous causes d'attente). Un cycle est l'expression du fait qu'une transaction A est en attente d'une transaction B qui est en attente d'une transaction A.

La détection d'un tel cycle permet de choisir une **victime**, c'est à dire une des deux transactions qui sera annulée pour que l'autre puisse se terminer.

Synonymes : Circuit de blocage

Définition : Délai d'attente

Principe de décision qu'une transaction doit être abandonnée (ROLLBACK) lorsque son délai d'attente est trop long.

Ce principe permet d'éviter les situations d'inter-blocage, en annulant une des deux transactions en cause, et en permettant donc à l'autre de se terminer.

Synonymes : *Timeout*

Remarque : Risque lié à l'annulation sur délai

Si le délai est trop court, il y a un risque d'annuler des transactions en situation d'attente longue, mais non bloquées.

Si le délai est trop long, il y a un risque de chute des performances en situation d'inter-blocage (le temps que le système réagisse).

La détection de cycle est plus adaptée dans tous les cas, mais plus complexe à mettre en œuvre.

Remarque : Relance automatique

Une transaction ayant été annulée suite à un inter-blocage (détection de cycle ou de délai d'attente) n'a pas commis de "faute" justifiant son annulation. Cette dernière est juste due aux contraintes de la gestion de la concurrence. Aussi elle n'aurait pas dû être annulée et devra donc être exécutée à nouveau.

Certains SGBD✯ se charge de relancer automatiquement les transactions ainsi annulées.

e) Mini-TP : Simulation d'accès concurrents sous PostgreSQL

1. Se connecter deux fois à une même base de données dans deux terminaux (**term1** et **term2**) :
   - `psql mydb`
   - `psql mydb`
2. Créer une table `test` : `CREATE TABLE test (a integer);`

Question 1

1. **term1** Insérer une valeur : `INSERT ...` (`COMMIT` implicite)
2. **term2** Vérifier que la valeur est visible : `SELECT ...`

Question 2

1. **term1** Commencer une transaction : `BEGIN TRANSACTION;`
2. **term1** Insérer la valeur 2 dans la table : `INSERT INTO...`
3. **term1** Vérifier que la valeur est bien dans la table : `SELECT * FROM ...`
4. **term2** Vérifier que la valeur **n'est pas visible** : `SELECT * FROM ...`
5. **term1** Valider la transaction : `COMMIT;`
6. **term2** Vérifier que la valeur **est visible** : `SELECT * FROM ...`

Question 3

1. **term1** Commencer une transaction : `BEGIN TRANSACTION;`
2. **term1** Exécuter une mise à jour (multiplier toutes les valeurs par 2) : `UPDATE...`
3. **term2** Exécuter une mise à jour concurrente (multiplier les valeurs par 3) : `UPDATE...`
4. **term2** Observer la mise en attente
5. **term1** Valider la transaction : `COMMIT;`

6. **term2** Vérifier le déblocage et que les deux mises à jour ont bien été effectuées (a multiplié par 6) : `SELECT...`
7. **term1** Vérifier que les deux mises à jour ont bien été effectuées (a multiplié par 6) : `SELECT...`

f) Solution aux trois problèmes soulevés par la concurrence.

Introduction

Le principe du verrouillage permet d'apporter une solution aux trois problèmes classiques soulevés par les accès aux concurrents aux données par les transactions.

i Perte de mise à jour

Temps	Transaction A	Transaction B
t1	LIRE T Verrou S	
t2	...	LIRE T Verrou S
t3	UPDATE T Attente...	...
t4	... Attente...	UPDATE T Attente...
...	Inter-blocage	

Tableau 16 Problème de la perte de mise à jour du tuple T par la transaction A

Remarque

Le problème de perte de mise à jour est réglé, mais soulève ici un autre problème, celui de **l'inter-blocage**.

ii Accès à des données non validées

Temps	Transaction A	Transaction B
t1		UPDATE T Verrou X
t2	LIRE T Attente...	...
t3		ROLLBACK Libération du verrou X
t4	Verrou S	

Tableau 17 Problème de la lecture impropre du tuple T par la transaction A

iii Lecture incohérente

Temps	Transaction A	Transaction B
t1	LIRE T Verrou S	
t2	...	UPDATE T Attente...
t3	LIRE T Verrous S	...
...	... libération des verrous ...	... reprise de la transaction ...

Tableau 18 Problème de la lecture non reproductible du tuple T par la transaction A

Remarque

La lecture reste cohérente car aucune mise à jour ne peut intervenir pendant le processus de lecture d'une même transaction.

g) Exercice

Soit l'exécution concurrente des deux transactions ci-dessous (on pose qu'aucune autre transaction ne s'exécute par ailleurs).

NB : Le tableau fait état des temps auxquels les instructions sont soumises au serveur et non auxquels elles sont traitées.

Temps	Transaction 1	Transaction 2
t0		BEGIN TRANSACTION
t1	BEGIN TRANSACTION	
t2	UPDATE TABLE1 SET TABLE1.A=TABLE1.A+1	
t3		UPDATE TABLE1 SET TABLE1.A=TABLE1.A+1
t4	UPDATE TABLE1 SET TABLE1.A=TABLE1.A+1	
t5	COMMIT	
t6		**?**

Tableau 19 Transactions concurrentes

De combien le champ TABLE1.A a-t-il été augmenté à t6 du point de vue de la transaction 2 ?

NB : Si vous pensez que le résultat ne peut être déterminé à cause d'une perte de mise à jour ou d'un inter-blocage, répondez 0.

h) Complément : Protocole d'accès aux données.

Méthode : Règles de verrouillage avant les lectures et écritures des données

Soit la transaction A voulant lire des données d'un tuple T :

1. A demande à poser un verrou S sur T
2. Si A obtient de poser le verrou alors A lit T
3. Sinon A attend le droit de poser son verrou (et donc que les verrous qui l'en empêchent soient levés)

Soit la transaction A voulant écrire des données d'un tuple T :

1. A demande à poser un verrou X sur T
2. Si A obtient de poser le verrou alors A écrit T
3. Sinon A attend le droit de poser son verrou (et donc que les verrous qui l'en empêchent soient levés)

Soit la transaction A se terminant (COMMIT ou ROLLBACK) :

1. A libère tous les verrous qu'elle avait posé
2. Certaines transactions en attente obtiennent éventuellement le droit de poser des verrous

Remarque : Liste d'attente

Afin de rationaliser les attentes des transactions, des stratégies du type FIFO★ sont généralement appliquées et donc les transactions sont empilées selon leur ordre de demande.

5. Synthèse : Les transactions

Transaction

Unité logique de travail pour assurer la cohérence de la BD même en cas de pannes ou d'accès concurrents.

- Panne

  Même en cas de panne, la BD doit rester cohérente.
  - Défaillances système
    Coupure de courant, de réseau, etc.
  - Défaillances du support
    Crash disque (dans ce cas les transactions peuvent être insuffisantes).
- Concurrence

  Dimension relevant de la conception d'application.
  - Perte de mise à jour
  - Accès à des données non valides
  - Lecture incohérente
- Programmation

  Un programme peut décider de l'annulation d'une transaction.
  - ROLLBACK
    Instruction SQL d'annulation d'une transaction.

6. Bibliographie commentée sur les transactions

Complément : Synthèses

Les transactions [w_developpez.com/hcesbronlavau]

Une bonne introduction courte au principe des transactions avec un exemple très bien choisi. Des exemples d'implémentation sous divers SGBD (InterBase par exemple)

SQL2 SQL3, applications à Oracle [Delmal01]

Une bonne description des principes des transactions, avec les exemples caractéristiques, l'implémentation SQL et une étude de cas sous Oracle 8 (chapitre 5).

Tutoriel de bases de données relationnelles de l'INT Evry [w_int-evry.fr] (*http://www-inf.int-evry.fr/COURS/BD/BD_REL/SUPPORT/poly.html#RTFToC30*[6]

Un aperçu général de l'ensemble de la problématique des transactions, de la concurrence et de la fiabilité.

Programmation SQL [Mata03]

Un exemple d'exécution de transactions (pages 20-23)

B. Exercices

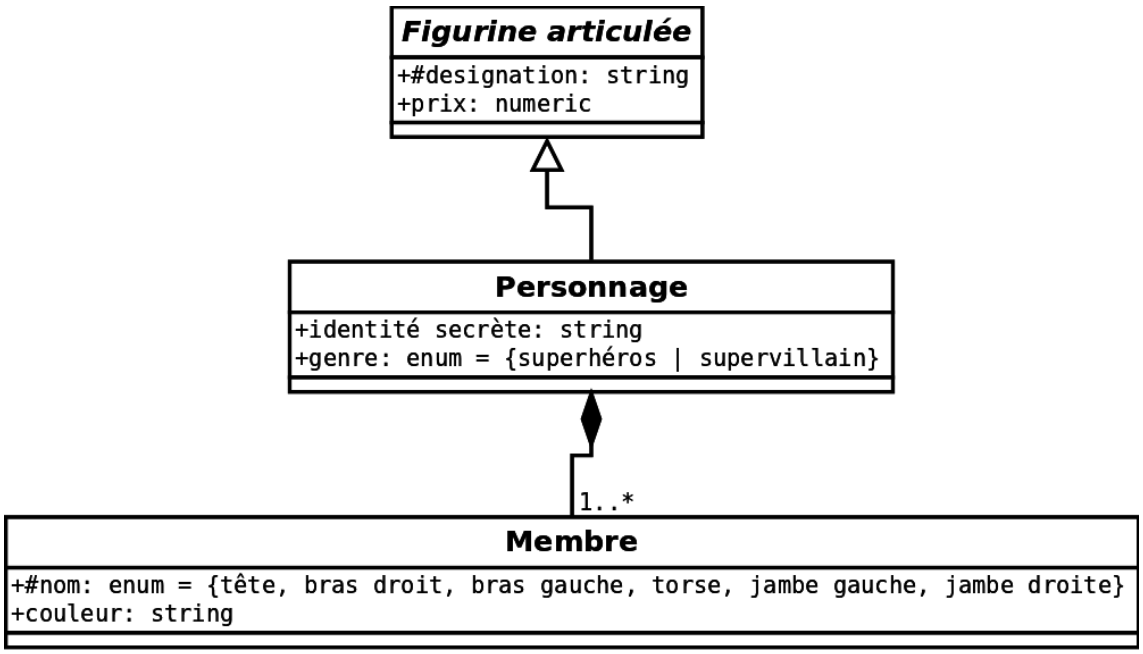
1. Super-héros sans tête

[10 minutes]

Les usines GARVEL construisent des figurines de super-héros à partir des données présentes dans la base de données PostgreSQL de l'entreprise. Un gros problème est survenu le mois dernier, lorsque l'usine en charge d'une nouvelle figurine, Superchild, a livré un million d'exemplaires sans tête. À l'analyse de la base il a en effet été observé que la base contenait un tuple "Superchild" dans la table Personnage, et cinq tuples associés dans la table Membre, deux pour les bras, deux pour les jambes et un pour le torse, mais aucun pour la tête.

Le service qui a opéré la saisie du nouveau personnage assure, sans ambiguïté possible, que la tête a pourtant été saisie dans la base. En revanche, l'enquête montre des instabilités de son réseau à cette période.

L'extrait du modèle UML utile au problème est proposé ci-après, ainsi que le code SQL exécuté via le client `psql` lors de l'insertion de la nouvelle figurine.



Modèle UML Figurines GARVEL (extrait)

```
1  \set AUTOCOMMIT on
```

6 - http://www-inf.int-evry.fr/COURS/BD/BD_REL/SUPPORT/poly.html#RTFToC30

```
2 INSERT INTO Personnage (designation, prix, identite_secrete, genre) VALUES
  ('Superchild','12','Jordy','superhéros') ;
3 INSERT INTO Membre (propriétaire, nom, couleur) VALUES ('Superchild','bras
  droit','bleu') ;
4 INSERT INTO Membre (propriétaire, nom, couleur) VALUES ('Superchild','bras
  gauche','bleu') ;
5 INSERT INTO Membre (propriétaire, nom, couleur) VALUES ('Superchild','jambe
  gauche','bleu') ;
6 INSERT INTO Membre (propriétaire, nom, couleur) VALUES ('Superchild','jambe
  droite','bleu') ;
7 INSERT INTO Membre (propriétaire, nom, couleur) VALUES
  ('Superchild','torse','rouge') ;
8 INSERT INTO Membre (propriétaire, nom, couleur) VALUES
  ('Superchild','tete','rose') ;
```

Question 1

Expliquer la nature du problème qui est probablement survenu. Proposer une solution générale pour que le problème ne se renouvelle pas, en expliquant pourquoi.

Soyez **concis** et **précis** : La bonne mobilisation des concepts du domaine et la clarté de la rédaction seront appréciées.

Question 2

Illustrer la solution proposée en corrigeant le code SQL de l'insertion de "Superchild".

2. Exercice : Films en concurrence

Soit la table Film suivante définie en relationnel permettant d'enregistrer le nombre d'entrées des films identifiés par leur ISAN.

```
1 Film(#isan:char(33),entrees:integer)
```

Soit l'exécution concurrente de deux transactions TR1 et TR2 visant à ajouter chacune une entrée au film '123' :

Temps	Transaction TR1	Transaction TR2
t0	BEGIN	
t1		BEGIN
t2	UPDATE Film SET entrees=entrees+1 WHERE isan='123'	
t3		
t4		UPDATE Film SET entrees=entrees+1 WHERE isan='123'
t5		
t6	COMMIT	
t7		
t8		COMMIT

Tableau 20 Transaction parallèles TR1 et TR2 sous PostgreSQL

NB :

- *Les instructions sont reportées au moment où elles sont transmises au serveur*
- *Aucune autre transaction n'est en cours d'exécution entre t0 et t8.*

Exercice

*De combien les entrées du film 123 ont-t-elles été augmentées à **t3** du point de vue de la transaction **TR1** ?*

Exercice

*De combien les entrées du film 123 ont-t-elles été augmentées à **t3** du point de vue de la transaction **TR2** ?*

Exercice

*De combien les entrées du film 123 ont-t-elles été augmentées à **t5** du point de vue de la transaction **TR1** ?*

Exercice

*De combien les entrées du film 123 ont-t-elles été augmentées à **t5** du point de vue de la transaction **TR2** ?*

Exercice

*De combien les entrées du film 123 ont-t-elles été augmentées à **t7** du point de vue de la transaction **TR1** ?*

Exercice

*De combien les entrées du film 123 ont-t-elles été augmentées à **t7** du point de vue de la transaction **TR2** ?*

Introduction à l'optimisation des bases de données

X

A. Cours

La conception des SGBDR exige qu'une attention particulière soit portée à la modélisation conceptuelle, afin de parvenir à définir des modèles logiques relationnels cohérents et manipulables. De tels modèles relationnels, grâce au langage standard SQL, présentent la particularité d'être implémentables sur toute plate-forme technologique indépendamment de considérations physiques.

Néanmoins l'on sait que dans la réalité, il est toujours nécessaire de prendre en considération les caractéristiques propres de chaque SGBDR, en particulier afin d'**optimiser** l'implémentation. Les optimisations concernent en particulier la question des **performances**, question centrale dans les applications de bases de données, qui, puisqu'elles manipulent des volumes de données importants, risquent de conduire à des temps de traitement de ces données trop longs par rapport aux besoins d'usage.

Chaque SGBDR propose généralement des mécaniques propres pour optimiser les implémentations, et il est alors nécessaire d'acquérir les compétences particulières propres à ces systèmes pour en maîtriser les arcanes. Il existe néanmoins des principes généraux, que l'on retrouvera dans tous les systèmes, comme par exemple les index, les groupements ou les vues matérialisées. Nous nous proposerons d'aborder rapidement ces solutions pour en examiner les principes dans le cadre de ce cours.

Nous aborderons également quelques techniques de conception, qui consistent à revenir sur la structure proposée par l'étape de modélisation logique, pour établir des modèles de données plus aptes à répondre correctement à des questions de performance. La dénormalisation ou le partitionnement en sont des exemples.

1. Introduction à l'optimisation du schéma interne

Objectifs

Assimiler la problématique de la performance en bases de données

Connaître les grandes classes de solutions technologiques existantes aux problèmes de performance

Connaître et savoir mobiliser les techniques de conception permettant d'optimiser les performances d'une BD

a) Schéma interne et performances des applications

La passage au schéma interne (ou physique), i.e. l'implémentation du schéma logique dans un SGBD✯ particulier, dépend de considérations pratiques liées aux performances des applications.

Les possibilités d'optimisation des schémas internes des BD✯ dépendent essentiellement des fonctions offertes par chaque SGBD.

On peut néanmoins extraire certains principes d'optimisation des schémas internes suffisamment généraux pour être applicables dans la plupart des cas.

Proposition de solutions

Parmi les solutions d'optimisation existantes, on pourra citer :

- L'indexation
- La dénormalisation
- Le partitionnement vertical et horizontal
- Les vues concrètes
- Le regroupement (*clustering*) de tables
- ...

Méthode : Démarche d'optimisation

1. Des problèmes de performance sont identifiés,
2. des solutions d'optimisation sont proposées,
3. les solutions sont évaluées pour vérifier leur impact et leur réponse au problème posé.

b) Évaluation des besoins de performance

Éléments à surveiller

Pour optimiser un schéma interne, il est d'abord nécessaire de repérer les aspects du schéma logique qui sont susceptibles de générer des problèmes de performance.

On pourra citer à titre d'exemple les paramètres à surveiller suivants :

- Taille des tuples
- Nombre de tuples
- Fréquence d'exécution de requêtes
- Complexité des requêtes exécutées (nombre de jointures, etc.)
- Fréquence des mises à jour (variabilité des données)
- Accès concurrents
- Distribution dans la journée des accès
- Qualité de service particulière recherchée
- Paramétrabilité des exécutions

- etc.

Évaluation des coûts

Une fois les éléments de la BD✯ à évaluer repérés, il faut mesurer si oui ou non ils risquent de poser un problème de performance.

L'évaluation des performances peut se faire :

- **Théoriquement**

  En calculant le coût d'une opération (en temps, ressources mémoires, etc.) en fonction de paramètres (volume de données, disparité des données, etc.). En général en BD le nombre de paramètres est très grand, et les calculs de coûts trop complexes pour répondre de façon précise aux questions posées.

- **Empiriquement**

  En réalisant des implémentations de parties de schéma et en les soumettant à des tests de charge, en faisant varier des paramètres. Ce modèle d'évaluation est plus réaliste que le calcul théorique. Il faut néanmoins faire attention à ce que les simplifications d'implémentation faites pour les tests soient sans influence sur ceux-ci.

c) Indexation

Définition : Index

Un index est une structure de données qui permet d'accélérer les recherches dans une table en associant à une clé d'index (la liste des attributs indexés) l'emplacement physique de l'enregistrement sur le disque.

Les accès effectuées sur un index peuvent donc se faire sur des structures optimisées pour la recherche (liste triée, B-tree...) au lieu de se faire par parcours séquentiel et intégral des enregistrements.

Exemple

Consulter une vidéo montrant la construction d'un index B-tree sur : *http://www.youtube.com/embed/coRJrcIYbF4*[7]

Cf. "Exemple de construction d'un B-tree (b-tree-example)"

Exemple de construction d'un B-tree

Méthode : Cas d'usage des index de type B-Tree

Les index doivent être utilisés sur les tables qui sont fréquemment soumises à des recherches. Ils sont d'autant plus pertinents que les requêtes sélectionnent un petit nombre d'enregistrements (moins de 25% par exemple).

Les index doivent être utilisés sur les attributs :

- souvent mobilisés dans une restriction (donc une jointure)
- très discriminés (c'est à dire pour lesquels peu d'enregistrements ont les mêmes valeurs)
- rarement modifiés

Attention : Inconvénients des index

- **Les index diminuent les performances en mise à jour (puisqu'il faut mettre à jour les index en même temps que les données).**
- **Les index ajoutent du volume à la base de données et leur volume peut devenir non négligeable.**

7 - http://www.youtube.com/embed/coRJrcIYbF4

Syntaxe : Créer un index en SQL

```
1  CREATE INDEX nom_index ON nom_table (NomColonneClé1, NomColonneClé2, ...);
```

Remarque : Index implicites

La plupart des SGBD✯ créent un index pour chaque clé (primaire ou candidate). En effet la vérification de la contrainte d'unicité à chaque mise à jour des données justifie à elle seule la présence de l'index.

Attention : Attributs indexés utilisés dans des fonctions

Lorsque les attributs sont utilisés dans des restrictions ou des tris par l'intermédiaire de fonctions, l'index n'est généralement pas pris en compte. L'opération ne sera alors pas optimisée. Ainsi par exemple dans le code suivant, le restriction sur X ne sera pas optimisée même s'il existe un index sur X.

```
1  SELECT *
2  FROM T
3  WHERE ABS(X) > 100
```

Cette non prise en compte de l'index est logique dans la mesure où, on le voit sur l'exemple précédent, une fois l'attribut soumis à une fonction, le classement dans l'index n'a plus forcément de sens (l'ordre des X, n'est pas l'ordre des valeurs absolues de X).

Lorsqu'un soucis d'optimisation se pose, on cherchera alors à sortir les attributs indexés des fonctions.

Notons que certains SGBD, tels qu'Oracle à partir de la version 8i, offrent des instructions d'indexation sur les fonctions qui permettent une gestion de l'optimisation dans ce cas particulier SQL2 SQL3, applications à Oracle [Delmal01].

d) Exercice

Soit les deux tables créées par les instructions suivantes :

```
1  CREATE TABLE T2 (
2    E char(10) Primary Key);
3
4  CREATE TABLE T1 (
5    A number(5) Primary Key,
6    B number(2) Not Null,
7    C char(10) References T2(E),
8    D char(5));
```

Soit une requête dont on cherche à optimiser les performances :

```
1  SELECT A
2  FROM T1, T2
3  WHERE C=E AND Abs(B)>50
4  ORDER BY D;
```

Sachant que tous les attributs sont très discriminés (c'est à dire que les enregistrements contiennent souvent des valeurs différentes les uns par rapport aux autres) et que les deux tables contiennent un grand nombre d'enregistrements, quelles sont les instructions de création d'index qui vont permettre d'optimiser l'exécution de cette requête ?

☐	CREATE INDEX idxA ON T1(A);
☐	CREATE INDEX idxB ON T1(B);
☐	CREATE INDEX idxC ON T1(C);
☐	CREATE INDEX idxD ON T1(D);
☐	CREATE INDEX idxE ON T2(E);

e) Dénormalisation

Rappel

La normalisation est le processus qui permet d'optimiser un modèle logique afin de le rendre non redondant. Ce processus conduit à la fragmentation des données dans plusieurs tables.

Définition : Dénormalisation

Processus consistant à regrouper plusieurs tables liées par des références, en une seule table, en réalisant statiquement les opérations de jointure adéquates.

L'objectif de la dénormalisation est d'améliorer les performances de la BD✯ en recherche sur les tables considérées, en implémentant les jointures plutôt qu'en les calculant.

Remarque : Dénormalisation et redondance

La dénormalisation est par définition facteur de redondance. A ce titre elle doit être utilisée à bon escient et des moyens doivent être mis en œuvre pour contrôler la redondance créée.

Méthode : Quand utiliser la dénormalisation ?

Un schéma doit être dénormalisé lorsque les performances de certaines recherches sont insuffisantes et que cette insuffisance à pour cause des jointures.

Attention : Inconvénients de la dénormalisation

La dénormalisation peut également avoir un effet néfaste sur les performances :

- **En mise à jour**

  Les données redondantes devant être dupliquées plusieurs fois.
- **En contrôle supplémentaire**

  Les moyens de contrôle ajoutés (*triggers*, niveaux applicatifs, etc.) peuvent être très coûteux.
- **En recherche ciblée**

  Certaines recherches portant avant normalisation sur une "petite" table et portant après sur une "grande" table peuvent être moins performantes après qu'avant.

Fondamental: Redondance et bases de données

La redondance volontaire est autorisée dans une base de données à trois conditions :

1. **avoir une bonne raison d'introduire de la redondance (améliorer des performances dans le cas de la dénormalisation)**
2. **documenter la redondance en explicitant les DF responsables de la non 3NF**
3. **contrôler la redondance par des mécanismes logiciels (triggers par exemple)**

f) Les trois principes à respecter pour introduire de la redondance dans une base de données

Fondamental

La redondance volontaire est autorisée dans une base de données à condition de respecter trois principes :

1. **il faut avoir une bonne raison d'introduire de la redondance (améliorer des performances dans le cas de la dénormalisation)**
2. **il faut documenter la redondance en explicitant les DF★ responsables de l'absence de 3NF**
3. **il faut contrôler la redondance par des mécanismes logiciels qui vont assurer que la redondance n'introduit pas d'incohérence (*triggers* par exemple)**

g) Exercice

Soit les deux tables créées par les instructions suivantes :

```
1  CREATE TABLE T2 (
2    C char(10) Primary Key,
3    E char(10));
4
5  CREATE TABLE T1 (
6    A char(10) Primary Key,
7    B char(10),
8    C char(10) References T2(C),
9    D char(10));
```

Parmi les instructions suivantes qui viendraient remplacer les deux créations précédentes, lesquelles implémenteraient une dénormalisation ?

☐
```
CREATE TABLE T3 (
B char(10) Primary Key,
D char(10));
CREATE TABLE T2 (
C char(10) Primary Key,
E char(10));
CREATE TABLE T1 (
A char(10) Primary Key,
B char(10) References T3(B),
C char(10) References T2(C));
```

☐
```
CREATE TABLE T1 (
A char(10) Primary Key,
B char(10),
C char(10) Unique Not Null,
D char(10),
E char(10));
```

☐
```
CREATE TABLE T1 (
A char(10) Primary Key,
B char(10),
C char(10),
D char(10),
E char(10));
```

☐
```
CREATE TABLE T5 (
E char(10) Primary Key);

CREATE TABLE T4 (
D char(10) Primary Key);

CREATE TABLE T3 (
B char(10) Primary Key);

CREATE TABLE T2 (
C char(10) Primary Key,
E char(10) References T5(E));

CREATE TABLE T1 (
A char(10) Primary Key,
B char(10) References T3(B),
C char(10) References T2(C),
D char(10) References T4(D));
```

h) Partitionnement de table

Définition : Partitionnement de table

Le partitionnement d'une table consiste à découper cette table afin qu'elle soit moins volumineuse, permettant ainsi d'optimiser certains traitements sur cette table.

On distingue :

- Le partitionnement vertical, qui permet de découper une table en plusieurs tables, chacune ne possédant qu'une partie des attributs de la table initiale.
- Le partitionnement horizontal, qui permet de découper une table en plusieurs tables, chacune ne possédant qu'une partie des enregistrements de la table initiale.

Définition : Partitionnement vertical

Le partitionnement vertical est une technique consistant à **implémenter des projections** d'une table T sur des tables T1, T2, etc. en répétant la clé de T dans chaque Ti pour pouvoir recomposer la table initiale par **jointure** sur la clé (Bases de données : objet et relationnel [Gardarin99]).

Remarque

Ce découpage équivaut à considérer l'entité à diviser comme un ensemble d'entités reliées par des associations 1:1.

Méthode : Cas d'usage du partitionnement vertical

Un tel découpage permet d'isoler des attributs peu utilisés d'autres très utilisés, et ainsi améliore les performances lorsque l'on travaille avec les attributs très utilisés (la table étant plus petite).

Cette technique diminue les performances des opérations portant sur des attributs ayant été répartis sur des tables différentes (une opération de jointure étant à présent requise).

Le partitionnement vertical est bien entendu sans intérêt sur les tables comportant peu d'attributs.

Définition : Partitionnement horizontal

Technique consistant à diviser une table T **selon des critères de restriction** en plusieurs tables T1, T2... et de telle façon que tous les tuples de T soit conservés. La table T est alors recomposable par **union** sur les Ti (Bases de données : objet et relationnel [Gardarin99]).

Méthode : Cas d'usage

Un tel découpage permet d'isoler des enregistrements peu utilisés d'autres très utilisés, et ainsi améliore les performances lorsque l'on travaille avec les enregistrements très utilisés (la table étant plus petite). C'est le cas typique de l'archivage. Un autre critère d'usage est le fait que les enregistrements soient toujours utilisés selon un partitionnement donné (par exemple le mois de l'année).

Cette technique diminue les performances des opérations portant sur des enregistrements ayant été répartis sur des tables différentes (une opération d'union étant à présent requise).

Le partitionnement horizontal est bien entendu sans intérêt sur les tables comportant peu d'enregistrements.

i) Exercice

Soit la table suivante contenant des listes de références produits :

```
1  CREATE TABLE Tref (
2    ref1 char(100) Primary Key,
3    ref2 char(100) Unique Not Null,
4    ref3 char(100) Unique Not Null,
5    ref4 char(100) Unique Not Null,
```

```
6    ref5 char(100) Unique Not Null,
7    ref6 char(100) Unique Not Null,
8    ref7 char(100) Unique Not Null);
```

On cherche à optimiser la requête suivante ("_" et "%" sont des jokers qui signifient respectivement "1 caractère quelconque" et "0 à N caractères quelconques") :

```
1  SELECT ref1
2  FROM Tref
3  WHERE (ref2 like 'REF42%' or ref2 like 'REF__7%' or ref2 like 'REF_78%');
```

Quelles sont les opérations de partitionnement qui permettront d'optimiser cette requête de façon maximale (indépendamment des conséquences pour d'éventuelles autres requêtes) ?

- ☐ Un partitionnement horizontal
- ☐ Un partitionnement vertical

j) Vues concrètes

Un moyen de traiter le problème des requêtes dont les temps de calcul sont très longs et les fréquences de mise à jour faible est l'utilisation de vues concrètes.

Définition : Vue concrète

Une vue concrète est un stockage statique (dans une table) d'un résultat de requête.

Un accès à une vue concrète permet donc d'éviter de recalculer la requête et est donc aussi rapide qu'un accès à une table isolée. Il suppose par contre que les données n'ont pas été modifiées (ou que leur modification est sans conséquence) entre le moment où la vue a été calculée et le moment où elle est consultée.

Une vue concrète est généralement recalculée régulièrement soit en fonction d'un événement particulier (une mise à jour par exemple), soit en fonction d'un moment de la journée ou elle n'est pas consultée et où les ressources de calcul sont disponibles (typiquement la nuit).

Synonymes : Vue matérialisée

Complément : Voir aussi

Vue matérialisée sous Oracle

k) Complément : Groupement de tables

Définition : Groupement de table

Un groupement ou *cluster* est une structure **physique** utilisée pour stocker des tables sur lesquelles doivent être effectuées de nombreuses requêtes comprenant des opérations de jointure.

Dans un groupement les enregistrements de plusieurs tables ayant une même valeur de champs servant à une jointure (clé du groupement) sont stockées dans un même bloc physique de la mémoire permanente.

Cette technique optimise donc les opérations de jointure, en permettant de remonter les tuples joints par un seul accès disque.

Définition : Clé du groupement

Ensemble d'attributs précisant la jointure que le groupement doit optimiser.

Syntaxe : Syntaxe sous Oracle

Il faut d'abord définir un groupement, ainsi que les colonnes (avec leur domaine), qui constituent sa clé. On peut ainsi ensuite, lors de la création d'une table préciser que certaines de ses colonnes appartiennent au groupement.

```
1  CREATE CLUSTER nom_cluster (NomColonneClé1 type, NomColonneClé2 type, ...);
2
3  CREATE TABLE nom_table (...)
4    CLUSTER nom_cluster (Colonne1, Colonne2, ...);
```

Remarque

Le *clustering* diminue les performances des requêtes portant sur chaque table prise de façon isolée, puisque les enregistrements de chaque table sont stockés de façon éclatée.

Remarque : Groupement et dénormalisation

Le groupement est une alternative à la dénormalisation, qui permet d'optimiser les jointures sans créer de redondance.

2. Mécanismes d'optimisation des moteurs de requêtes

Objectifs

Comprendre les principes de fonctionnement du moteur d'optimisation interne aux SGBD

a) Calcul du coût d'une requête

Soit deux relations `t1(a:char(8),b:integer)` et `t2(b:integer)` de respectivement 100 et 10 lignes.

Soit la requête `SELECT * FROM t1 INNER JOIN t2 ON t1.b=t2.b WHERE t2.b=1`.

- Soit n1 le nombre de tuples tels que b=1 dans t1 ; n1 ∈[0..100]
- Soit n2 le nombre de tuples tels que b=1 dans t2 ; n2 ∈[0..10]
- Soit n3 le nombre de tuples tels que t1.b=t2.b ; : n3 ∈[0..1000]

Question 1

Il existe plusieurs façons de traduire cette requête en algèbre relationnel, proposer quelques exemples.

Question 2

Calculer le nombre d'instructions de chaque exemple d'expression relationnelle :

- en considérant que les restrictions sont effectuées par un parcours intégral dans des listes non triées (comme si c'était effectué avec des boucles de type FOR) ;
- en considérant que chaque opération est effectuée séparément.

Effectuer les calculs pour trois valeurs de n1,n2 et n3 (une valeur minimum, une valeur maximum et une valeur intermédiaire). Donner un intervalle [min..max] indépendant de n1, n2 et n3.

Question 3

Conclure en indiquant quelle opération nécessite le moins d'instructions.

b) Principe de l'optimisation de requêtes

L'exécution d'une requête SQL suit les étapes suivantes :

1. Analyse syntaxique : Vérification de la correction syntaxique et traduction en opérations algébriques
2. Contrôle : Vérification des droits d'accès
3. **Optimisation** : Génération de plans d'exécution et choix du meilleur

4. Exécution : Compilation et exécution du plan choisi

Définition : Plan d'exécution

L'analyse de la requête permet de produire un **arbre d'opérations** à exécuter.

Or il est possible de transformer cet arbre pour en obtenir d'autres équivalents, qui proposent des **moyens différents pour arriver au même résultat**, on parle de différents plans d'exécution.

Définition : Optimisation de la requête

L'optimisation de la requête est une opération informatique visant à réécrire l'arbre d'exécution de la requête en vue de choisir le plan d'exécution le plus performant.

c) Techniques pour l'optimisation de requêtes

Restructuration algébrique

Il existe des propriétés permettant de modifier l'ordre des opérateurs algébriques, afin d'obtenir des gains de performance par exemple :

- Le groupage des restrictions : Il permet d'effectuer plusieurs restrictions en un seul parcours de la table, au lieu d'un par restriction

  `Restriction (Restriction (t,x=a), x=b) <=> Restriction (t,x=a ET x=b)`
- La commutativité des restrictions et Jointures : Elle permet d'appliquer les restrictions avant les jointures
- L'associativité des jointures : Elle permet de changer l'ordre des jointures, afin d'utiliser des algorithmes plus performants dans certains cas
- ...

Heuristiques d'optimisation

Le moteur d'optimisation de la base de données va poser des règles de réécriture pour produire des arbres d'exécutions et choisir les moins coûteux.

Il va typiquement **appliquer d'abord les opérations réductrices** (restrictions et projections).

informations statistiques (modèle de coût)

Une des actions du moteur est d'ordonner les jointures (et les opérations ensemblistes) afin de :

- traiter le moins de tuples possibles ;
- mobiliser les index existants au maximum ;
- utiliser les algorithmes les plus performants.

Pour cela le moteur d'optimisation a besoin de connaître le contenu de la BD : nombre de tuples dans les tables, présence ou non d'index, ...

Le moteur de la BD met à sa disposition des informations statistiques répondant à ces besoins.

d) Collecte de statistiques pour le moteur d'optimisation

Le moteur d'optimisation d'une BD a besoin de collecter des informations sur les tables qu'il a à manipuler afin de choisir les meilleurs plans d'exécution, sur la taille des tables typiquement.

Il est nécessaire de commander au moteur de collecter ces statistiques lorsque des changements de contenu importants ont été effectués dans la BD.

Syntaxe : PostgreSQL : Analyse d'une table

```
1 ANALYSE <table> ;
```

Syntaxe : PostgreSQL : Analyse de toute la base

```
1 ANALYSE;
```

Syntaxe : PostgreSQL : Récupération des espaces inutilisés et analyse de toute la base

```
1 VACUUM ANALYZE ;
```

Complément : Exemple de données collectées

- nombre de lignes
- volume des données
- valeurs moyennes, minimales, maximales
- nombre de valeurs distinctes (indice de dispersion)
- nombre d'occurrences des valeurs les plus fréquentes (notamment pour les colonnes pourvues d'index...)

http://sqlpro.developpez.com/cours/sqlaz/fondements/#L2.1[8]

e) Jointures et optimisation de requêtes

Algorithmes

Il existe plusieurs algorithmes pour réaliser une jointure :

- Jointures par boucles imbriquées (*nested loop*) : chaque tuple de la première table est comparé à chaque tuple de la seconde table.
- Jointure par tri-fusion (*sort-merge*) : les deux tables sont d'abord triées sur l'attribut de jointure, puis elles sont fusionnées.
- Jointure par hachage (*hash join*) : les deux tables sont hachées, puis jointes par fragment.

3. Analyse et optimisation manuelle des requêtes

Objectifs

Savoir afficher et interpréter un plan d'exécution de requête
Savoir écrire des requêtes performantes

a) Analyse de coûts de requêtes (EXPLAIN)

Définition : Plan d'exécution

Le moteur SQL d'une BD peut afficher le plan qu'il prévoit d'exécuter pour une requête donnée, c'est à dire la liste des opération qui vont être exécutées, ainsi qu'une estimation du coup de ces opérations.

Syntaxe : Syntaxe PostgreSQL

```
1 EXPLAIN <requête SQL>
```

8 - http://sqlpro.developpez.com/cours/sqlaz/fondements/#L2.1

Exemple

Soit la table "taero" des aérodromes de France.

- `SELECT count(*) FROM taero;`
  retourne 421
- `SELECT * FROM taero LIMIT 7;`
  retourne la table ci-après.

	code character(4)	nom character varying(50)
1	LFAB	DIEPPE ST AUBIN
2	LFAC	CALAIS DUNKERQUE
3	LFAD	COMPIEGNE MARGNY
4	LFAE	EU MERS LE TREPORT
5	LFAF	LAON CHAMBRY
6	LFAG	PERONNE SAINT QUENTIN
7	LFAI	NANGIS LES LOGES

Image 37 Sept premières lignes de la table "taero"

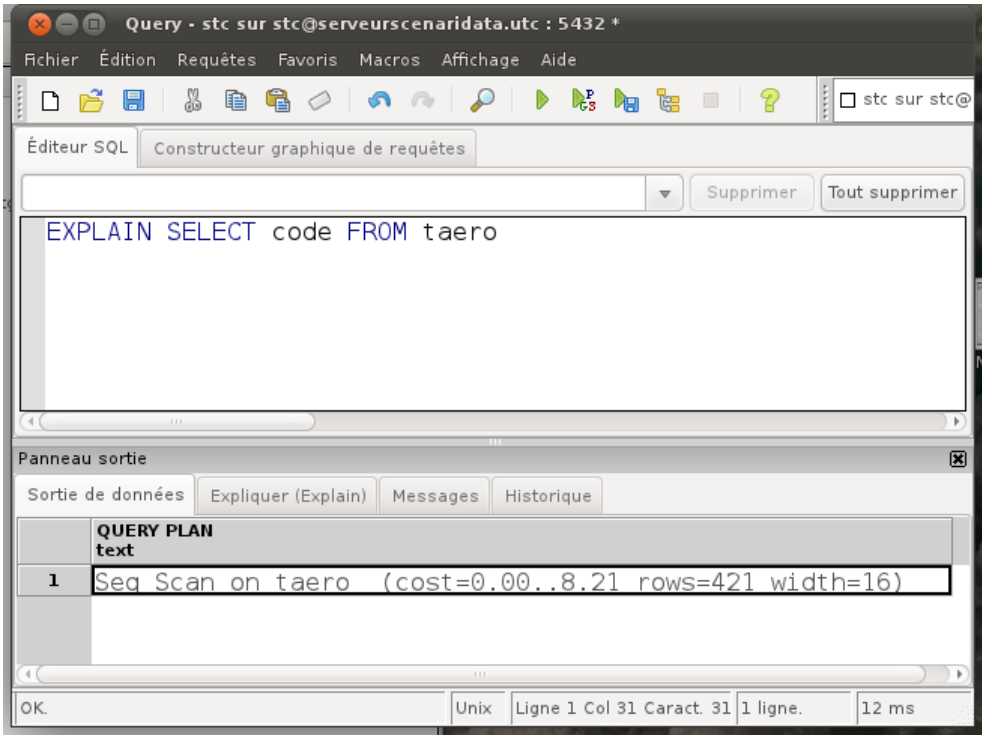


Image 38 EXPLAIN SELECT code FROM taero

`EXPLAIN SELECT code FROM taero` retourne `"Seq Scan on taero (cost=0.00..8.21 rows=421 width=16)"`

- L'opération exécute un `sequential scan` (parcours séquentiel de toute la table)
- Le **coût estimé de démarrage** (coût pour récupérer le premier enregistrement) est de 0.00 et le **coût estimé total** (coût pour récupérer tous les tuples) est de 8.21.
  L'unité est relative aux pages disque accédées, mais elle peut être considérée arbitrairement pour faire des comparaisons.
- Le **nombre estimé de lignes** rapportées est de 421, pour une **taille** de 16 octets chacune.

Définition : Analyse d'exécution

Le moteur SQL d'une BD peut exécuter une requête et afficher le coût réellement observé d'exécution des opérations.

Syntaxe : Syntaxe PostgreSQL

```
1 EXPLAIN ANALYSE <requête SQL>
```

Exemple

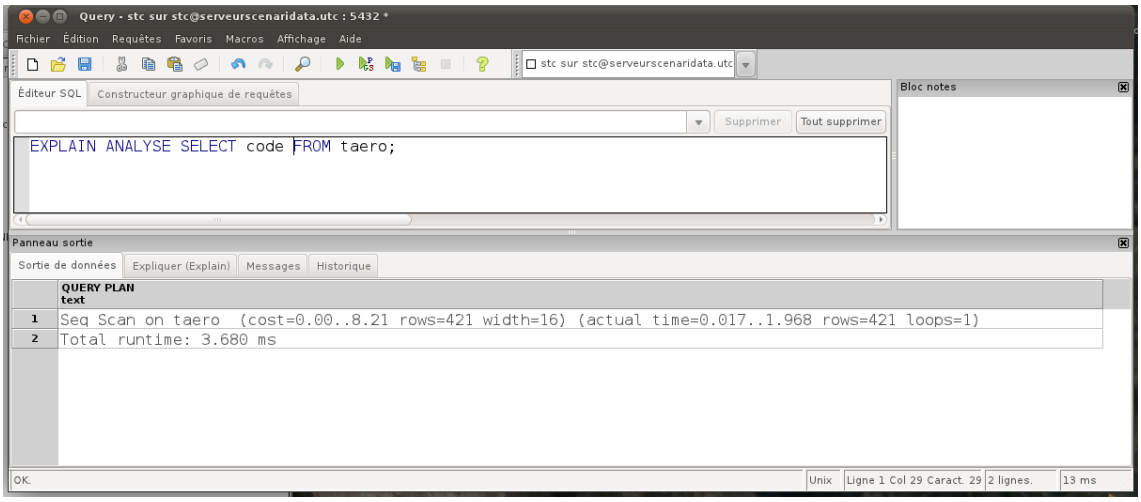


Image 39 EXPLAIN ANALYSE SELECT code FROM taero

`EXPLAIN ANALYSE SELECT code FROM taero` retourne `"Seq Scan on taero (cost=0.00..8.21 rows=421 width=16) (actual time=0.017..1.960 rows=421 loops=1) Total runtime: 3.668 ms"`

- Le **coût réel de démarrage** de l'opération est de 0.017ms et le **coût réel total** est de 1.960ms
- Le **nombre de lignes** réellement rapporté est de 421
- `loops=1` indique l'opération de `Seq Scan` a été exécuté une seule fois
- Le coût total de 3.668ms inclut le temps lié à l'environnement du moteur d'exécution (*start and stop*).

Attention

1. **EXPLAIN ne donne qu'une valeur estimée algorithmiquement du coût :**
   - **cette estimation est indépendante de contingences matériel, c'est un coût théorique**
   - **cette estimation est reproductible (chaque exécution donne des informations identiques)**
   - **l'instruction SQL n'est pas exécutée**
2. **ANALYSE donne un temps de calcul mesuré réellement :**
   - **ce temps mesuré dépend de l'environnement matériel (état du réseau, charge du serveur, ...)**
   - **ce temps mesuré est variable à chaque exécution (en fonction des éléments précédents)**
   - **l'instruction SQL est exécutée**
     (on peut insérer la requête à analyser dans une transaction et effectuer un ROLLBACK si l'on souhaite analyser l'exécution d'une requête sans l'exécuter)

Remarque

`EXPLAIN` n'existe pas dans le standard SQL.

Complément : Pour aller plus loin : PostgreSQL

- *http://www.postgresql.org/docs/current/static/sql-explain.html*[9]
- *http://www.postgresql.org/docs/current/static/using-explain*[10]
- *http://www.bortzmeyer.org/explain-postgresql.html*[11]

Syntaxe : Syntaxe sous Oracle

```
1 EXPLAIN PLAN FOR <requête SQL>
```

Il faut préalablement avoir créé ou avoir accès à une table particulière PLAN_TABLE qui sert à stocker les résultats du EXPLAIN.

Complément : Pour aller plus loin : Oracle

- *http://jpg.developpez.com/oracle/tuning/*[12]

b) Optimisation manuelle des requêtes

La façon d'écrire des requêtes influe sur l'exécution. L'idée générale est d'écrire des requêtes qui :

- minimisent les informations retournées (par exemple en faisant toutes les projections possibles) ;
- minimisent les traitements (par exemple en faisant toutes les restrictions possibles) ;
- évitent des opérations inutiles (par exemple : `SELECT DISTINCT`, `ORDER BY`...) ;

Complément : Trucs et astuces...

http://sqlpro.developpez.com/cours/optimiser/#L9[13]

4. Synthèse : L'optimisation

Optimisation

Modification du schéma interne d'une BD pour en améliorer les performances.

- Techniques au niveau physique
  - Indexation
  - Regroupement physique
  - Vue concrète
- Techniques de modélisation
  - Dénormalisation
  - Partitionnement
    - Horizontal
    - Vertical

5. Bibliographie commentée sur l'optimisation

Complément : Synthèses

SQL2 SQL3, applications à Oracle [Delmal01]

9 - http://www.postgresql.org/docs/current/static/sql-explain.html
10 - http://www.postgresql.org/docs/current/static/using-explain
11 - http://www.bortzmeyer.org/explain-postgresql.html
12 - http://jpg.developpez.com/oracle/tuning/
13 - http://sqlpro.developpez.com/cours/optimiser/#L9

Une bonne description des index (page 79) et clusters (page 81) par l'image. Une description intéressante d'une **fonction** (sous Oracle 8i) qui permet d'indexer des fonctions, ce qui répond à un problème important d'optimisation (page 84). Une description des vues matérialisées, leur implémentation sous Oracle, des exemples d'usage dans le domaine du *datawarehouse*.

Bases de Données Relationnelles et Systèmes de Gestion de Bases de Données Relationnels, le Langage SQL [w_supelec.fr/~yb]

Des informations générales sur les *index*[14] et sur les *clusters*[15].

B. Exercices

1. Film concret

Soit le schéma relationnel suivant :

```
1  Film (#isan:char(33), titre:varchar(25), entrees:integer,
     nomReal=>Realisateur(nom), prenomReal=>Realisateur(prenom))
2  Realisateur (#nom:varchar(25), #prenom:varchar(25), ddn:date)
```

Soit la requête suivante portant sur ce schéma implémenté sous PostgreSQL :

```
1  SELECT f.titre AS film, r.ddn AS real
2  FROM Film f, Realisateur r
3  WHERE f.nomReal=r.nom AND f.prenomReal=r.prenom
```

Question

Proposer une optimisation de cette requête sous la forme de la vue matérialisée `vTopFilms`.

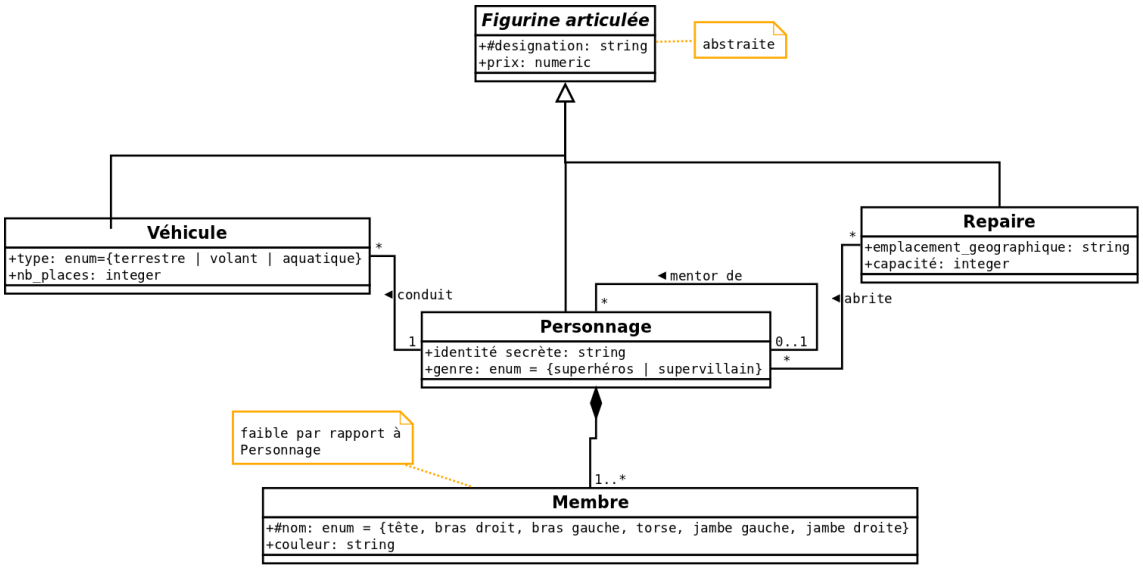
2. Super-lents

[10 minutes]

L'entreprise GARVEL propose des figurines de super-héros à acheter en ligne sur un site Internet adossé à sa base de données. Son catalogue a atteint cette année le million de modèles. Depuis quelques temps, et la récente forte augmentation des modèles au catalogue, les performances des consultations ont beaucoup chuté, entraînant des temps d'accès lents (plusieurs secondes) et une baisse des actes d'achat.

La requête la plus utilisée par l'application Web sert à lister tous les super-héros avec toutes leurs caractéristiques, avec leurs véhicules et leurs repaires (et également toutes leurs caractéristiques) et à la trier par ordre de prix.

14 - http://wwwsi.supelec.fr/~yb/poly_bd/node122.html
15 - http://wwwsi.supelec.fr/~yb/poly_bd/node126.html



Modèle UML Figurines GARVEL

Soyez **concis** et **précis** dans vos réponses ; La bonne mobilisation des concepts du domaine et la clarté de la rédaction seront appréciées.

Question 1

Expliquer pourquoi cette requête peut poser des problèmes de performance, lorsque la base comprend de nombreux enregistrements.

Question 2

Proposer et justifier une première solution d'optimisation à mettre en œuvre qui sera utile dans tous les cas et n'aura que peu d'effets indésirables. Puis, expliquer pourquoi le passage en relationnel-objet de la base de données, combiné à la solution précédente, peut améliorer considérablement les performances.

Question 3

Proposer deux solutions alternatives qui, si l'on reste en relationnel, permettraient d'améliorer considérablement les performances. Vous en poserez les avantages, inconvénients et les mesures à prendre pour contenir ces inconvénients.

Glossaire

Relation toute clé

En base de données, on appelle une relation toute clé une relation dont tous les attributs sont nécessaires pour constituer une clé.

Signification des abréviations

- 1NF First Normal Form
- 2NF Second Normal Form
- 3NF Third Normal Form
- 4NF Fourth Normal Form
- 5NF Fifth Normal Form
- ACID Atomicity, Consistency, Isolation, Durability
- BCNF Boyce-Codd Normal Form
- BD Base de Données
- DDN Date De Naissance
- DF Dépendance Fonctionnelle
- DFE Dépendance Fonctionnelle Elémentaire
- E-A Entité-Association
- FIFO First In First Out
- LCD Langage de Contrôle de Données
- LMD Langage de Manipulation de Données
- MCD Modèle Conceptuel de Données
- MLD Modèle Logique de Données
- SGBD Système de Gestion de Bases de Données
- SQL Structured Query Language
- UML Unified Modeling Language

Bibliographie

[Celko00] Celko Joe. *SQL avancé : Programmation et techniques avancées*. Vuibert, 2000.

[Delmal01] Delmal Pierre. *SQL2 SQL3, applications à Oracle*. De Boeck Université, 2001.

[Gardarin99] Gardarin Georges. *Bases de données : objet et relationnel*. Eyrolles, 1999.

[Mata03] Mata-Toledo Ramon A., Cushman Pauline K.. *Programmation SQL*. Ediscience, 2003.

[Roques04] Roques Pascal, Vallée Franck. *UML 2 en action : De l'analyse des besoins à la conception J2EE*. ISBN 2-212-11462-1 (3ème édition). Paris : Eyrolles, 2004. 385 p. architecte logiciel.

[Roques09] Pascal Roques, *UML 2 par la pratique*, 7e édition, Eyrolles, 2009 [ISBN 978-2212125658]

Webographie

[w_developpez.com/hcesbronlavau] Cesbron-Lavau Henry, *Les transactions*, *http://www.developpez.com/hcesbronlavau/Transactions.htm* , 2002.

[w_dia] *Dia*, *http://live.gnome.org/Dia*

[w_int-evry.fr] Defude Bruno, *Tutoriel de bases de données relationnelles de l'INT Evry*, *http://www-inf.int-evry.fr/COURS/BD/* , consulté en 2009.

[w_journaldunet.com(1)] Morlon Jérôme, *UML en 5 étapes*, *http://developpeur.journaldunet.com/dossiers/alg_uml.shtml* , 2004.

[w_journaldunet.com(2)] Borderie Xavier, *Cinq petits conseils pour un schéma UML efficace*, *http://developpeur.journaldunet.com/tutoriel/cpt/031013cpt_uml5conseils.shtml* , 2004.

[w_objecteering] *Objecteering software. http://www.objecteering.com* . [2002-septembre].

[w_supelec.fr/~yb] Bourda Yolaine, *Bases de Données Relationnelles et Systèmes de Gestion de Bases de Données Relationnels, le Langage SQL*, *http://wwwsi.supelec.fr/~yb/poly_bd/* , consulté en 2009.

[w_uml.free.fr] *UML en Français*, *http://uml.free.fr* , consulté en 2002.

Index