Modélisation des données

3IF – INSA Lyon

Jean-Marc Petit

2014-2015

Sur ce cours

Volume horaire :	9 cours, 5 TD, 1 examen final

Commentaires/suggestions :
`jean-marc.petit@insa-lyon.fr`

Personnes ayant contribué à ce support (ordre alphabétique) :
Marie Agier, Fabien De Marchi, Frédéric Flouvat, Hélène Jaudoin, Farouk Toumani

Enfin et surtout : participez !

Objectifs du cours

- Donner des bases solides pour comprendre la gestion des données
- Focus sur les bases de données relationnelles : structure, interrogation, conception

Deux "gros pavés" autour des

1. Langages
2. Contraintes

Ce que ce cours fait peu, voire pas du tout

1. Les Systèmes de Gestion de Bases de Données (SGBD) et leur mise en œuvre (SQL en pratique)
   $\Rightarrow$ Voir le cours de Philippe Lamarre sur SQL, très complémentaire de ce cours
2. Présenter une méthode de conception de système d'information (SI) type MERISE, RUP ...
3. Les aspects fonctionnels, dynamiques et organisationnels des SI,

Références bibliographiques

- Mark Levene, Georges Loizou, “Guided tour of relational databases and beyond", 625 pages, 1999, Springer
- Serge Abiteboul, Rick Hull, Victor Vianu, “Foundations of databases”, 685 pages, 1995 (*existe aussi en Français*), Addison-Wesley
- Heikki Mannila, Kari-Jouko Räihä, “The Design of Relational Databases”, Second edition, 1994, Addison-Wesley

Sommaire

Outline

Le modèle relationnel

1. Structure
   Donne l'organisation des données du modèle, i.e. sous forme de tableaux à deux dimensions (LDD).
2. Langages
   Il existe trois principaux langages de requêtes relationnels :
   - Algèbre relationnelle
   - Calcul relationnel
   - Datalog Non-recursif : langage à base de règles

   Ce sont des langages déclaratifs qui ont donné lieu au langage SQL utilisé en pratique.
3. Contraintes : basées sur différents types de dépendances
   Restreindre les relations autorisées dans une base de données pour satisfaire certaines conditions logiques, appelées *contraintes d'intégrité*.

Outline

Structure du modèle relationnel

Les données sont structurées en **relation** (ou *table* ou tableau à deux dimensions)

- les colonnes donnent les caractéristiques de ce que la relation représente
- les lignes donnent les "individus" qui peuplent la relation

Exemple

Personnes	*nss*	*nom*	*prenom*	*age*
	12	*Aymard*	*Serge*	*45*
	45	*Fenouil*	*Solange*	*35*

Içi, la première ligne donne les noms aux différentes caractéristiques, appelés *attributs*.
$\Rightarrow$ Version dite **nommée** du modèle relationnel

Représentation équivalente de la relation Personnes

$\{<$ 12, Aymard, Serge, 60 $>$, $<$45, Fenouil, Solange, 55) $>\}$

{ Personnes(nss : 12, nom : Aymard, prenom : Serge, age : 60), Personnes(nss : 45, nom : Fenouil, prenom : Solange, age : 55)}

{ Personnes(12, Aymard, Serge, 60), Personnes(45, Fenouil, Solange, 55) }

$\Rightarrow$ On utilisera par la suite ces différentes représentations (nommées et non-nommées), mais en attendant, on se focalise sur la **version nommée** de la diapo précédente.

Attributs et domaines

Soit $\mathcal{U}$ un ensemble infini dénombrable de noms d'attributs ou **attributs**, nommé **univers**.

Exemple

$\mathcal{U} = \{nss, nom, prenom, age, dep, adresse, activite\}$

Soit $\mathcal{D}$ un ensemble infini dénombrable de *valeurs constantes*. $\mathcal{D}$ représente l'ensemble des valeurs qui peuvent être prises dans une base de données.

Pour $A \in \mathcal{U}$, on note $dom(A)$ le **domaine** de A, i.e. le sous ensemble de $\mathcal{D}$ dans lequel A peut prendre des valeurs ($dom(A) \subseteq \mathcal{D}$).

Exemple

$dom(dep) = \{Math, Info, Phys, Bio, Chimie\}$

Symbole de relation

Un **symbole de relation** est un symbole R auquel on associe :

- un entier naturel $type(R)$, qui donne le nombre d'attributs de R,
- une fonction att_R définie de $\{1, \ldots, type(R)\}$ dans $\mathcal{U}$ qui associe des attributs de $\mathcal{U}$ à R dans un certain ordre.

On note $schema(R) = \{att_R(1), \ldots, att_R(type(R))\}$ l'ensemble des attributs de R, i.e. le *schéma* de R.

Remarque : par abus de langage, on confond parfois le symbole avec le schéma de relation.

Exemple

Soit Personne un symbole de relation avec $type(Personne) = 4$ *et* $schema(Personne) = \{att(1), att(2), att(3), att(4)\}$ *où* $att(1) = nss$, $att(2) = nom$, $att(3) = prenom$, *et* $att(4) = age$

Formes normales

Caractérisation de certaines formes de schémas de relations

Definition

Un symbole de relation R est en *première forme normale (1FN)* si tous les domaines des attributs de R ont des valeurs constantes et **atomiques**.

Hypothèse relativement forte

Tuples et relations

Soit R un symbole de relation avec
$schema(R) = \{A_1, \ldots, A_m\}, att_R(i) = A_i, i = 1, m$
Un **tuple** sur R est un élément du produit cartésien
$dom(A_1) \times \ldots \times dom(A_m)$
Une **relation** sur R est un ensemble fini de tuples sur R.

Exemple

La relation Personnes (exemple 1) est une relation sur le symbole de relation Personne.
$< 12, Aymard, Serge, 45 > \in Personnes$

Domaine actif

Le **domaine actif d'un attribut** $A \in schema(R)$ dans r, noté $ADOM(A, r)$, est l'ensemble des valeurs constantes prises par A dans r, i.e. $ADOM(A, r) = \pi_A(r)$.

Le **domaine actif d'une relation** r, noté ADOM(r), est défini par $ADOM(r) = \bigcup_{A \in schema(R)} ADOM(A, r)$.

$\Rightarrow$ à ne pas confondre avec le domaine d'un attribut

Manipulation sur un tuple

Soient R un symbole de relation avec $schema(R) = \{A_1, \ldots, A_m\}$, $att_R(i) = A_i (i = 1, m)$, r une relation sur R, t un tuple de r et $A_i \in schema(R)$.

La **projection** d'un tuple t sur A_i, notée $t[A_i]$, est la ième coordonnée de t, i.e. t(i).

$\rightarrow$ se généralise à plusieurs attributs
Soit $X = \{att_R(i_1), \ldots, att_R(i_n)\} \subseteq schema(R)$. La projection de t sur X, notée $t[X]$, est définie par $t[X] =< t(i_1), \ldots, t(i_n) >$

Exemple

Soit $t =< 12, Aymard, Serge, 45 >$. *On a :*
$t[nom] = t[att(2)] = t(2) =< Aymard >$
$t[nom, prenom] = t[att(2), att(3)] =< t(2), t(3) >=<$
$Aymard, Serge >$

Symbole de base de données

Un **symbole de base de données R** est un ensemble de symboles de relation.

Soit $\mathbf{R} = \{R_1, \ldots, R_n\}$. On a :

- $schema(\mathbf{R}) = \bigcup_{R_i \in \mathbf{R}} schema(R_i)$
- Un symbole de BD est en 1FN si ses symboles de relation sont en 1FN.

Hypothèses de nommage

Hypothèse implicite du modèle relationnel
Universal Relation Schema Assumption (URSA) : Si un attribut apparaît dans plusieurs schémas de relation, alors cet attribut représente les mêmes données.

Exemple

Soit **R** = { *Personne, Departement, Travaille* } *avec schema(Personne)={ nss, nom, prenom, age }, schema(Departement)={ dep, adresse }, schema(Travaille)={ nss, dep, activite }.*

Base de données

Une **base de données d** est définie sur un symbole de base de données et représente un ensemble de relations.

Soit $\mathbf{R} = \{R_1, \ldots, R_n\}$. $\mathbf{d} = \{r_1, \ldots, r_n\}$ est une base de données sur $\mathbf{R}$ r_i définie sur R_i (i=1,n)

Le **domaine actif d'une base de données d** , noté ADOM(**d**), est l'union des domaines actifs de ses relations.

Exemple

Soit $\boldsymbol{d} = \{$ *Personnes, Departements, Travaillent* $\}$ *avec définie sur Personne, Departement, Travaille respectivement.*

Personnes	*nss*	*nom*	*prenom*	*age*
	12	*Aymard*	*Serge*	*45*
	45	*Fenouil*	*Solange*	*35*

Departements	*dep*	*adresse*
	Math	*Carnot*
	Info	*Cézeaux*

Travaillent	*nss*	*dep*	*activite*
	12	*Math*	*Prof*
	45	*Math*	*MdC*
	45	*Info*	*MdC*

Outline

Langages de requêtes

Langages dits *déclaratifs* puisqu'il suffit de **décrire** le résultat recherché sans spécifier **comment** l'obtenir.

- **Approche algébrique** : Algèbre relationnelle
  Très utile pour représenter les plans d'exécution des requêtes puisque un *ordre* pour l'exécution de la requête est donnée.
  $\Rightarrow$ dotée d'une *sémantique opérationnelle*
- **Approche logique**
  - Calcul relationnel
    Orienté utilisateur car il est plus proche du langage naturel : on ne spécifie pas d'ordre.
    $\Rightarrow$ dotée d'une *sémantique déclarative*
  - Datalog
    Langage déclaratif à base de règles, capacités d'inférence

$\Rightarrow$ SQL = sucre syntaxique de ces langages

Focus sur les requêtes conjonctives

Proposée par Codd en 1970

- collection d'*opérateurs algébriques*
  - Entrée : une ou deux relations
  - Sortie : une relation
- Possibilité de *composition* des opérateurs

La projection

Soient r une relation sur R et $Y \subseteq \mathit{schema}(R)$.
La **projection** de r sur Y, notée $\pi_Y(r)$, est définie par :

$$\pi_Y(r) = \{t[Y] \mid t \in r\}$$

Soit S le symbole de relation associé à $\pi_Y(r)$. On a $\mathit{schema}(S) = Y$

$\Rightarrow$ correspond à une *coupe verticale* de la relation

La sélection (1/2)

Nécessite tout d'abord de définir la notion de **formule de sélection** sur R, puis la **satisfaction d'une formule de sélection par un tuple**.

Définition d'une formule de sélection sur R par induction, i.e. formule simple puis formule de sélection

- Une *formule simple* sur R est une expression de la forme : $A = a$ ou $A = B$, où $A, B \in \mathit{schema}(R)$ et $a \in \mathit{dom}(A)$
- Une *formule de sélection* est
  - soit une formule simple,
  - soit une expression de la forme $F_1 \wedge F_2, F_1 \vee F_2, \neg F_1$ ou (F_1) avec F_1, F_2 des formules de sélection.

Note : on peut étendre ces formules à d'autres opérateurs binaires : $<, >, \leq, \ldots$

La sélection (2/2)

Soient r une relation sur R, $t \in r$ et F une formule de sélection sur R

- t satisfait F, noté $t \models F$, est défini inductivement par :
  1. $t \models A = a$ si $t[A] = a$
  2. $t \models A = B$ si $t[A] = t[B]$
  3. $t \models F_1 \wedge F_2$ si $t \models F_1$ **et** $t \models F_2$
  4. $t \models F_1 \vee F_2$ si $t \models F_1$ **ou** $t \models F_2$
  5. $t \models \neg F$ si $t \not\models F$
  6. $t \models (F)$ si $t \models F$

Soient r une relation sur R et F une formule de sélection sur R. La **sélection** des tuples de r par rapport à F, notée $\sigma_F(r)$, est définie par :

$$\sigma_F(r) = \{t \in r \mid t \models F\}$$

$\Rightarrow$ correspond à une *coupe horizontale* de la relation

Opérations ensemblistes

Soient r_1 et r_2 deux relations sur le même symbole de relation R

- Union :

$$r_1 \cup r_2 = \{t \mid t \in r_1 \text{ ou } t \in r_2\}$$

- Différence :

$$r_1 - r_2 = \{t \mid t \in r_1 \text{ et } t \notin r_2\}$$

- Intersection :

$$r_1 \cap r_2 = \{t \mid t \in r_1 \text{ et } t \in r_2\}$$

Exemple

Donner $r_1 \cup r_2$, $r_1 \cap r_2$ *et* $r_1 - r_2$.

r_1

A	B	C
1	2	3
1	1	1
1	2	2

r_2

A	B	C
2	2	2
1	1	1

Quelques propriétés

- $r_1 \cap r_2 = r_1 - (r_1 - r_2)$
- $\sigma_{\neg F}(r) = r - \sigma_F(r)$
- $\sigma_{F_1 \vee F_2}(r) = \sigma_{F_1}(r) \cup \sigma_{F_2}(r)$
- $\sigma_{F_1 \wedge F_2}(r) = \sigma_{F_1}(r) \cap \sigma_{F_2}(r)$

La jointure naturelle

Soient r_1 et r_2 deux relations sur R_1 et R_2 respectivement.

La **jointure naturelle** de r_1 et r_2, notée $r_1 \bowtie r_2$, est une relation sur un symbole de relation R, avec
$schema(R) = schema(R_1) \cup schema(R_2)$, définie par :
$r_1 \bowtie r_2 =$
$\{t \mid \exists t_1 \in r_1 \exists t_2 \in r_2 \text{ tq } t[schema(R_1)] = t_1 \text{ et } t[schema(R_2)] = t_2\}$

Note : cette définition est "non constructive", elle caractérise la jointure sans donner d'indices sur la façon de la calculer en pratique.

Un algorithme de jointure

Algorithme Jointure(r_1, r_2)
Data: r_1, r_2 deux relations sur R_1, R_2 resp.
Result: *Res*, le résultat de la jointure
Soit $X = schema(R_1) \cap schema(R_2)$;
Resultat $= \emptyset$;
foreach $t_1 \in r_1$ **do**
    foreach $t_2 \in r_2$ **do**
        if $t_1[X] = t_2[X]$ **then**
            Construire un tuple t tq $t[schema(R_1)] = t_1$ et
            $t[schema(R_2)] = t_2$;
            Res = Res $\cup$ { t } ;
Return(Res) ;

Propriété

$Jointure(r_1, r_2) = r_1 \bowtie r_2$

$\Rightarrow$ Propriété constructive pour calculer la jointure

Exemples de jointure

Donner $s_1 \bowtie s_2$, $s_1 \bowtie s_3$, $s_1 \bowtie s_4$.

s_1

A	B	C
2	2	2
1	1	1

s_2

A	D	E
1	2	3
1	1	2
2	3	1
3	1	2

s_3

A	B	C
1	1	1
1	2	3

s_4

D	E	F
1	1	1
2	2	2

Lien entre la jointure, le produit cartésien et l'intersection

- Si $schema(R_1) = schema(R_2)$, que vaut $r_1 \bowtie r_2$?
- Si $schema(R_1) \cap schema(R_2) = \emptyset$, que vaut $r_1 \bowtie r_2$?

Le renommage

$\rightarrow$ permet par exemple de forcer ou d'éviter des jointures naturelles

Soit r une relation sur R, $A \in schema(R)$ et $B \notin schema(R)$

Le **renommage** de A en B dans r, noté $\rho_{A \rightarrow B}(r)$, est une relation sur S avec $schema(S) = (schema(R) - \{A\}) \cup \{B\}$ définie par :
$\rho_{A \rightarrow B}(r) =$
$\{t \mid$
$\exists u \in r,\ t[schema(S) - \{B\}] = u[schema(R) - \{A\}] \text{ et } t[B] = u[A]\}$

La division : l'intuition

$\Rightarrow$ capture la sémantique du **quantificateur universel** ($\forall$).

Example

*"Quels sont des étudiants qui sont inscrits dans **tous** les départements"*

Inscriptions

etud	dep
1	1
1	2
1	3
2	2

Departement

dep
1
2
3

Définition de la division

Soient r une relation sur R avec $schema(R) = XY$ et s une relation sur S avec $schema(S) = Y$.

La **division** de r par s, notée $r \div s$, est une relation sur un symbole de relation R_1, avec $schema(R_1) = X$, définie par :

$$r \div s = \{t[X] \mid t \in r \text{ et } s \subseteq \pi_Y(\sigma_F(r))\}$$

avec $X = \{A_1, \ldots, A_q\}$ et $F = (A_1 = t[A_1]) \wedge \ldots \wedge (A_q = t[A_q])$

Propriété

$r \div s = \pi_X(r) - \pi_X((\pi_X(r) \times s) - r)$

Exemples

Donner $r_1 \div r_2$, $r_1 \div r_3$ et $r_1 \div r_4$.

r_1

etud	dep
1	1
1	2
1	3
1	4
2	1
2	2
3	2
4	2
4	4

r_2

dep
2

r_3

etud
2
4

r_4

dep
1
2
4

Synthèse sur l'algèbre relationnelle

- Opérations ensemblistes
  - Union ($\cup$) : sélection des tuples d'une relation r_1 et de ceux d'une autre relation r_2
  - Intersection ($\cap$)
  - Différence ($-$) : sélection des tuples d'une relation r_1 qui n'appartiennent pas à une relation r_2
- Autres opérations
  - Projection (π) : suppression des attributs d'une relation
  - Sélection (σ) : sélection d'un sous ensemble de tuples d'une relation.
  - Jointure ($\bowtie$) : combine deux relations
  - Renommage ($\rho_{A \to B}$) : renomme le nom d'un attribut
  - Produit cartésien ($\times$)
  - Division ($\div$) : exprime le quantificateur universel ($\forall$)

Exercices

Certains opérateurs sont indispensables, d'autres pas, i.e. ils peuvent s'exprimer à partir des autres.

Par exemple, on peut exprimer

- une jointure $\bowtie$ avec $\times, \rho, \pi, \sigma$,
- un produit cartésien $\times$ avec $\bowtie, \rho$
- l'intersection $\cap$ avec $-$

Requêtes algébriques

Soit d une base de données

- Une *requête algébrique* sur d est une *expression* composée d'un nombre **fini** d'opérateurs algébriques sur les relations de d
- Un ordre d'évaluation des opérateurs est spécifié dans la requête
  $\Rightarrow$ à la base de l'optimisation de requêtes

Arbres de requêtes

Une expression algébrique peut se représenter sous forme d'**arbre** :

- la racine de l'arbre correspond à la requête
- les feuilles de l'arbre correspondent aux relations
- les noeuds de l'arbre correspondent aux opérateurs algébriques

Exercice

Représenter $Q = \pi_X(\sigma_{C_1}(r_1 \bowtie r_2) \bowtie (\sigma_{C_2}(r_3) \cup r_4))$

$\Rightarrow$ permet de mieux comprendre l'optimisation logique de requêtes

NB : IBM DB2 offre une représentation physique de cet arbre,
Oracle aussi mais de façon moins aisée.

Propriétés algébriques

Utiles pour la réécriture des requêtes

$\sigma_F(\sigma_{F'}(r)) = \sigma_{F'}(\sigma_F(r))$

$\sigma_F(\pi_X(r)) = \pi_X(\sigma_F(r))$ si F porte sur X

$\sigma_F(r \bowtie s) = \sigma_F(r) \bowtie s$ si F porte sur $schema(R)$

$\pi_X(r \bowtie s) = \pi_X(r) \bowtie \pi_X(s)$ si $schema(R) \cap schema(S) = X$

$(r_1 \cup r_2) \bowtie r_3 = (r_1 \bowtie r_3) \cup (r_2 \bowtie r_3)$

Complétude

Un langage d'interrogation de bases de données relationnelles est dit *relationnellement complet* si il peut exprimer toute requête exprimable dans l'algèbre relationnelle

Extensions de l'algèbre relationnelle

- Prise en compte des valeurs nulles
- Requêtes récursives (aussi appelé Fermeture transitive)
- Fonctions d'agrégation sur les données

Exemple de requêtes récursives

Famille

Parent	Enfant
p1	p3
p1	p4
p2	p4
p2	p5
p3	p6
p3	p7
p7	p12

"Donner la liste des parents et leurs petits-fils"

$\pi_{par,enf}(\rho_{Enfant \to AJ}(\rho_{Parent \to par}(Famille)) \bowtie \rho_{Parent \to AJ}(\rho_{Enfant \to enf}(Famille))))$

"Donner tous les descendants de p1 "

Requête impossible en algèbre relationnelle

$\Rightarrow$ Possible en SQL

clause "connect by" avec Oracle (DBMS specific)

Outline

Petit exemple introductif

Soit $R = \{Departement, Etudiant\}$ un schéma de BD avec $schema(Departement) = \{Loc, Mgr, Dept\}$ et $schema(Etudiant) = \{Name, Age, Address, Dept\}$ et soient *Departements* et *Etudiants* deux relations.

Etudiants

Name	Age	Address	Dept
Vidal	21	Clermont	Computing
Dupond	35	Paris	Maths
Durand	45	Lyon	Linguistics
Dan	34	Lyon	Computing

Departements

Loc	Mgr	Dept
Bât A	Ullman	Computing
Bât B	Mendelson	Maths
Bât C	Dupond	Linguistics

Langages logiques

Donner les noms et âges des étudiants du département géré par "Ullman".

Deux approches :

- Approche basée sur les `variables du domaine`
  Si les tuples $\langle x_1, x_2, x_3, x_4 \rangle$ et $\langle y_1, 'Ullman', y_2 \rangle$ sont respectivement dans les relations *Etudiants* et *Departments* et $x_4 = y_2$ alors inclure dans la réponse le tuple
  $\langle x_1 : Name, x_2 : Age \rangle$,
  où $x_1, x_2, x_3, x_4, y_1, y_2$ sont des variables de domaine
- Approche basée sur les `variables de tuple`
  Si les variables de tuple t_1 et t_2 définies respectivement sur *Etudiants* et *Departments* sont tels que le manager de t_2 est $'Ullman'$ et les noms de département (attribut *dept*) de t_1 et t_2 sont les mêmes alors inclure dans la réponse les nom et âge du tuple t_1.

Le calcul relationnel

- Deux types de calculs : calcul relationnel de domaine (DRC) et calcul relationnel de tuples (TRC).
- Le calcul utilise des variables, des constantes, des opérateurs de comparaison, des connecteurs logiques et des quantificateurs
  - DRC : les variables prennent comme valeurs des éléments du domaine.
  - TRC : les variables prennent comme valeurs des tuples.

  TRC et DRC sont des sous-ensembles de la logique du premier ordre
- Les expressions dans le calcul sont appelées *formules*
- Un tuple réponse est une affectation de constantes à des variables qui permet à la formule d'être évaluée à vrai

Syntaxe DRC

Symboles autorisés dans une formule

- Constantes : $v_1, \ldots, v_n$ membres de l'ensemble $\mathcal{D}$
- Variables : $x_1, \ldots, x_n$ membres d'un ensemble infini dénombrable $\mathcal{V}$ disjoint de $\mathcal{D}$ et de $\mathcal{U}$
- Symboles de relations : $R_1, R_2, \ldots, R_n$
- Opérateurs de comparaison : $=$
- Quantificateurs et connecteurs logiques : $\exists, \forall, \wedge, \vee, \neg$
- Délimiteurs : () et ,

Note : On peut admettre d'autres opérateurs $<, >, =, \leq, \geq, \neq$

Formules DRC

$\Rightarrow$ définies de façon inductive : formules atomiques, puis formules plus complexes utilisant des connecteurs logiques.

- Formule atomique
  - $R(x_1, \ldots, x_n)$ avec R symbole de relation, $type(R) = n$ et x_i une constante ou une variable pour tout $i \in \{1, \ldots, n\}$
  - $x = y$, avec x une variable et y une constante ou une variable
- Formule bien formée
  - Une formule atomique
  - $(p), \neg p, p \wedge q, p \vee q$ où p et q sont des formules bien formées
  - $\exists x : A(F), \forall x : A(F)$ où F est une formule bien formée, x une variable et A un nom d'attribut

Remarque sur la représentation structurelle

L'ordre des variables est important dans les formules atomiques de la forme $R(x_1, \ldots, x_n)$, i.e. x_i représente $att_R(i)$

Correspond toujours à une version nommée du modèle relationnel mais avec un ordre sur l'apparition des variables/constantes dans les prédicats (relations).

Variables libres et variables liées

Les quantificateurs $\exists$ et $\forall$ permettent de *lier* les variables

- Toutes les variables apparaissant dans une formule atomique sont libres
- Les variables libres apparaissant dans $\neg F$ sont les mêmes que celles apparaissant dans la formule F
- Les variables libres apparaissant dans les formules $F_1 \wedge F_2$ et $F_1 \vee F_2$ sont les variables libres qui apparaissent dans F_1 ou dans F_2
- Les variables libres qui apparaissent dans $\exists x : A(F)$ et $\forall x : A(F)$ sont les variables libres qui apparaissent dans F excepté les occurrences de x dans F

Requêtes en calcul relationnel de domaine

Soit $\mathbf{R} = \{R_1, \ldots, R_m\}$ un schéma de base de données.

- Une requête sur $\mathbf{R}$ à la forme suivante :

$$\{\langle x_1 : A_1, \ldots, x_n : A_n \rangle | F(\langle x_1, \ldots, x_n \rangle)\}$$

où

- F est une formule *bien formée*,
- $A_1, \ldots, A_n$ sont des attributs appartenant à $\mathcal{U}$
- $x_1, \ldots, x_n$ sont des variables libres dans F et 2 à 2 disjointes, les autres variables étant liées

$\Rightarrow$ Notion purement syntaxique

$\Rightarrow$ Reste à définir la sémantique d'une requête, i.e. son résultat

Notion d'interprétation

Une interprétation sur un schéma de base de données $\mathbf{R}$ est définie par

- une base de données d sur $\mathbf{R}$ et
- une affectation σ des variables $\mathcal{V}$ aux constantes de $\mathcal{D}$

L'affectation σ est une fonction de $\mathcal{V}$ dans $\mathcal{D}$
Soit x une variable sur le domaine de A. On note $\sigma : x \mapsto v$ avec $v \in DOM(A)$

On note $\langle d, \sigma \rangle$ une interprétation sur $\mathbf{R}$.

On notera aussi $\sigma_{x \mapsto v}$ l'affectation σ à laquelle on ajoute $\sigma(x) = v$

Satisfaction d'une formule

Soient une interprétation $\langle d, \sigma \rangle$ et une formule δ sur $\mathbf{R}$
La satisfaction de δ par rapport à $\langle d, \sigma \rangle$, notée $\langle d, \sigma \rangle \models \delta$, est définie inductivement comme suit :

- $\langle d, \sigma \rangle \models x = v$ ssi $\sigma(x) = v$
- $\langle d, \sigma \rangle \models x = y$ ssi $\sigma(x) = \sigma(y)$
- $\langle d, \sigma \rangle \models R(x_1, \ldots, x_n)$ ssi
  $\langle \sigma(x_1), \ldots, \sigma(x_n) \rangle \in r \quad (r \in d, r \text{ sur } R)$
- $\langle d, \sigma \rangle \models (F)$ ssi $\langle d, \sigma \rangle \models F$
- $\langle d, \sigma \rangle \models \neg F$ ssi $\langle d, \sigma \rangle \not\models F$
- $\langle d, \sigma \rangle \models F_1 \wedge F_2$ ssi $\langle d, \sigma \rangle \models F_1$ et $\langle d, \sigma \rangle \models F_2$
- $\langle d, \sigma \rangle \models F_1 \vee F_2$ ssi $\langle d, \sigma \rangle \models F_1$ ou $\langle d, \sigma \rangle \models F_2$
- $\langle d, \sigma \rangle \models \exists x : A(G)$ ssi $\exists v \in DOM(A)$ tel que $\langle d, \sigma_{x \mapsto v} \rangle \models G$
- $\langle d, \sigma \rangle \models \forall x : A(G)$ ssi $\forall v \in DOM(A), \langle d, \sigma_{x \mapsto v} \rangle \models G$

Résultat d'une requête

Soit $d = \{r_1, \ldots, r_m\}$ une base de données sur $\mathbf{R}$ et $Q = \{\langle x_1 : A_1, \ldots, x_n : A_n \rangle | F(\langle x_1, \ldots, x_n \rangle)\}$ une requête DRC sur $\mathbf{R}$

La réponse de Q sur d produit une relation $ans(Q, d)$ définie par :

$$ans(Q, d) = \{\langle \sigma(x_1), \ldots, \sigma(x_n) \rangle \mid \langle d, \sigma \rangle \models F\}$$

Le schéma de $ans(Q, d)$ est $\{A_1, \ldots, A_n\}$

Exemple

Soit $\mathbf{R} = \{Departement, Etudiant\}$ un schéma de BD avec $schema(Departement) = \{Loc, Mgr, Dept\}$ et $schema(Etudiant) = \{Name, Age, Address, Dept\}$ et soit $d = \{Departements, Etudiants\}$ une BD sur $\mathbf{R}$.

Etudiants

Name	Age	Address	Dept
Vidal	21	Clermont	Computing
Dupond	35	Paris	Maths
Durand	45	Lyon	Linguistics
Dan	34	Lyon	Computing

Departements

Loc	Mgr	Dept
Bât A	Ullman	Computing
Bât B	Mendelson	Maths
Bât C	Dupond	Linguistics

Exemples de requêtes

- "Donner la liste de tous les étudiants"
  $\{\langle x_1 : \mathit{Name}, x_2 : \mathit{Age}, x_3 : \mathit{Address}, x_4 : \mathit{Dept}\rangle \mid$
  $\quad \mathit{Etudiant}(x_1, x_2, x_3, x_4)\}$
- "Donner les noms et départements de tous les étudiants"
  $\{\langle x_1 : \mathit{Name}, x_4 : \mathit{Dept}\rangle \mid \exists x_2 : \mathit{Age}(\exists x_3 : \mathit{Address}($
  $\quad \mathit{Etudiant}(x_1, x_2, x_3, x_4)))\}$
- Donner le nom, l'age et le département des étudiants qui habitent à 'Clermont'
  $\{\langle x_1 : \mathit{Name}, x_2 : \mathit{Age}, x_4 : \mathit{Dept}\rangle \mid$
  $\quad \exists x_3 : \mathit{Address}(\mathit{Etudiant}(x_1, x_2, x_3, x_4) \wedge x_3 = \text{`}\mathit{Clermont}\text{`})\}$

Exemples de requêtes (cont.)

- "Donner les noms et âges des étudiants qui sont soit inscrits dans le département *Linguistics* soit habitent à *Clermont*"
  $\{\langle x_1 : Name, x_2 : Age\rangle \mid \exists x_3 : Address(\exists x_4 : Dept($
  $Etudiant(x_1, x_2, x_3, x_4) \wedge (x_4 = \text{`}Linguistics\text{`} \vee x_3 =$
  $\text{`}Clermont\text{`})))\}$
  ou $\{\langle x_1 : Name, x_2 : Age\rangle \mid \exists x_4 : Dept($
  $Etudiant(x_1, x_2, \text{`}Clermont\text{`}, x_4)) \vee$
  $\exists x_3 : Address(Etudiant(x_1, x_2, x_3, \text{`}Linguistics\text{`}))\}$
- "Donner les noms des étudiants non parisiens qui ne sont pas inscrits dans le département *Computing*"
  $\{\langle x_1 : Name\rangle \mid \exists x_2 : Age(\exists x_3 : Address(\exists x_4 : Dept($
  $Etudiant(x_1, x_2, x_3, x_4) \wedge (\neg(x_4 = \text{`}Computing\text{`}) \wedge \neg(x_3 =$
  $\text{`}Paris\text{`})))))\}$

Exemples de requêtes (cont.)

- "Donner les noms des étudiants et les localisations des départements dans lesquels ils sont inscrits"
  $Q = \{\langle x_1 : Name, x_5 : Loc\rangle \mid$
  $\exists x_2 : Age(\exists x_3 : Address(\exists x_4 : Dept(\exists x_6 : Mgr($
  $Etudiant(x_1, x_2, x_3, x_4) \wedge Department(x_5, x_6, x_4)))))\}$

Remarque : Toutes ces requêtes sont décrites sur le schéma de la BD uniquement. Pour les évaluer sur d, en prenant la requête Q ci dessus, on doit écrire $ans(Q, d)$.

Calcul relationnel autorisé : une restriction du calcul

Exemple de requêtes peu sûres : Requêtes de calcul

- correctes syntaxiquement
- . . . mais nombre infini de réponses **ou** réponses toujours vides

Exemple

Soit r une relation sur R avec $schema(R) = \{A\}, dom(A) = \mathcal{N}$ et $r = \{0, 1, 2\}$.
Soient les 2 requêtes suivantes :
$Q_1 : \{\langle x : A\rangle | R(x)\}$
$Q_2 : \{\langle x : A\rangle | \neg R(x)\}$

Pouvez vous écrire une requête équivalente en algèbre relationnelle ?

Notions préliminaires

Intimement lié à une notion d'*indépendance de domaine*
Comment se ramener au domaine actif uniquement ?

- *Domaine actif d'une BD d*
  noté $ADOM(d)$
- *Domaine actif d'une requête Q*
  noté $ADOM(Q)$, et égal à l'ensemble des constantes apparaissant dans Q
- *Indépendance du domaine*
  Une requête Q est indépendante du domaine si pour toute base de données d, les réponses de Q sur d dépendent uniquement de $ADOM(Q) \cup ADOM(d)$

Requêtes indépendantes du domaine

Les requêtes de l'algèbre relationnelle (et les requêtes Datalog sûres) sont indépendantes du domaine, *celles du calcul relationnels ne le sont pas*.

Exemple

Soient Etudiant et Inscrit deux schémas de relations avec $schema(Etudiant) = schema(Inscrit) = \{nom\}$. *Les requêtes suivantes ont une réponse infinie si* $DOM(nom)$ *est infini.*

$\{< x : nom > | \neg Etudiant(x)\}$

$\{< x_1 : nom, x_2 : nom > | Etudiant(x_1) \vee Inscrit(x_2)\}$

D'où deux problèmes principaux dans les requêtes du calcul :

- La requête contient une formule $\neg F$ telle qu'une des variables de F n'apparaît pas positivement ailleurs
- La requête contient une formule de la forme $F_1 \vee F_2$ telle que les variables libres de F_1 sont distinctes de celles de F_2

Requêtes indépendantes du domaine (suite)

Théorême

Déterminer si une requête du calcul relationnel est indépendante du domaine est indécidable.

Idée : restreindre les formules de façon **syntaxique** de façon à rendre les requêtes du calcul indépendante du domaine.

Deux notions :

- variables positives
- variables négatives

Variable positive

Une variable x est *positive* dans une formule si :

- x est positive dans une formule atomique $R(y_1, \ldots, x, \ldots, y_k)$
- x est positive dans une formule atomique $x = v$ où v est une constante
- x est positive dans la formule $\neg F$ qui contient x, si x est négative dans F (si F n'est pas une formule atomique, il faut "pousser" le constructeur $\neg$)
- x est positive dans la formule $F_1 \wedge F_2$ si x est positive dans F_1 ou x est positive dans F_2
- x est positive dans la formule $F_1 \vee F_2$ si x est positive dans F_1 et x est positive dans F_2
- x est positive dans la formule $\exists y : A(F)$, si $x \neq y$ et x est positive dans F

Variable négative

Une variable x est *négative* dans une formule si :

- x est négative dans une formule atomique $x = y$ où y est une variable
- x est négative dans la formule $\neg F$ si x est positive dans F
- x est négative dans la formule $F_1 \wedge F_2$ si x est négative dans F_1 et x est négative dans F_2
- x est négative dans la formule $F_1 \vee F_2$ si x est négative dans F_1 ou x est négative dans F_2
- x est négative dans la formule $\forall y : A(F)$, si $x \neq y$ et x est négative dans F
- si x n'apparaît pas dans une formule F, alors x est négative dans F

Exemple

Soient $Etudiant$, $Inscrit$ et $Habite$ trois schémas de relations avec $schema(Etudiant) = schema(Inscrit) = \{nom\}$ et $schema(Habite) = \{nom, adresse\}$.

Soient les formules suivantes :
$F_1 = Etudiant(x)$
$F_2 = Etudiant(x_1) \vee Inscrit(x_2)$
$F_3 = Habite(x_1, x_2) \wedge \neg Habite(x_1, Lyon)$
$F_4 = Habite(x_1, Antraigues) \vee Habite(x_1, Lyon)$

Donner le status des variables dans $F_1 - F_4$.

Formule du calcul autorisée

Une formule du calcul F est *autorisée* si :

- Chaque variable libre x de F est positive dans F
- Pour chaque sous-formule $\exists x : A(G)$ de F, x est positive dans G
- Pour chaque sous-formule $\forall x : A(G)$ de F, x est négative dans G

Requête de calcul relationnel autorisée

- Une requête autorisée à la forme suivante :

$$\{\langle x_1 : A_1, \ldots, x_n : A_n \rangle | F(\langle x_1, \ldots, x_n \rangle)\}$$

où : F est une formule *bien formée* et *autorisée*,
$A_1, \ldots, A_n$ sont des attributs appartenant à $\mathcal{U}$
$x_1, \ldots, x_n$ sont des variables libres dans F 2 à 2 disjointes, les autres variables étant liées

Tester si une requête du calcul est autorisée peut se faire par simple récursion sur la structure de la formule.

Théorême

Toute formule autorisée du calcul relationnel est indépendante du domaine.

Exemple (suite)

Soient *Etudiant*, *Inscrit* et *Habite* trois schémas de relations avec $schema(Etudiant) = schema(Inscrit) = \{nom\}$ et $schema(Habite) = \{nom, adresse\}$.

Les requêtes ci-dessous sont-elles autorisées ?

$Q_1 = \{< x : nom > | \neg Etudiant(x)\}$
$Q_2 = \{< x_1 : nom, x_2 : nom > | Etudiant(x_1) \vee Inscrit(x_2)\}$
$Q_3 = \{< x_1 : nom > | \exists x_2 : adresse(Habite(x_1, x_2) \wedge$
$\neg Habite(x_1, Lyon))\}$
$Q'_3 = \{< x_1 : nom > | \forall x_2 : adresse(Habite(x_1, x_2) \wedge$
$\neg Habite(x_1, Lyon))\}$

Exemple (fin)

$$Q_4 = \{< x_1 : nom > | Habite(x_1, Antraigues) \vee Habite(x_1, Lyon)\}$$
$$Q_5 = \{< x : nom > | \forall y : nom(\neg Etudiant(y))\}$$
$$Q'_5 = \{< x : nom > | \neg(\exists z : nom(Etudiant(z)))\}$$
$$Q_6 = \{< y : adresse > | \forall x : nom(\neg Etudiant(x) \wedge Habite(x, y)))\}$$

Outline

Datalog (Data Logic)

Langage de requêtes déclaratif à base de règles
$\Rightarrow$ Langage logique muni de capacité d'inférence

Langage plus puissant que l'algèbre relationnelle et le calcul relationnel

- récursivité permise

Syntaxe de Datalog : un sous ensemble de la syntaxe de Prolog
Expressivité ?

- Prolog est équivalent aux langages de programmation (i.e. Turing-complet)
- Datalog ne l'est pas.

Syntaxe de Datalog

$\Rightarrow$ Version non-nommée du modèle relationnel (plus d'attributs, uniquement l'ordre dans les prédicats compte)

- `Formule atomique` : les mêmes que celles du calcul relationnel de domaine
  Néanmoins, on nomme :
  - *Formule de prédicat* ou simplement prédicat, toute formule atomique de la forme $R(x_1, \ldots, x_n)$ (sensible à l'ordre)
  - *Formule d'égalité* ou simplement *égalité*, toute formule atomique de la forme $x = y$
  - Atome clos sur R, toute formule atomique de la forme $R(v_1, \ldots, v_n)$, chaque v_i étant une constante de $DOM(att(i))$
- Littéral : il est soit une formule atomique L (littéral positif) soit sa négation $\neg L$ (littéral négatif)

Notion de règle (ou clause)

Une `règle` est une expression de la forme :

$$\overbrace{L}^{Tete} \vdash \overbrace{L_1, \ldots, L_n}^{Corps}$$

où $L, L_1, \ldots, L_n$ sont des littéraux
Si $n = 0$ et L est un atome clos, alors L est un fait.

Exemple

$Parent(Paul, Jean)$
$Parent(Jean, Emma)$
$GParent(x, y) \vdash Parent(x, z), Parent(z, y)$

Une règle est une *"usine à fournir des faits"*

Programme Datalog

Definition

Un `programme Datalog` est un ensemble fini de règles $C_1, \ldots, C_m$

Deux types de programmes Datalog : récursifs et non recursifs
$\Rightarrow$ on se focalise sur les programmes Datalog non récursifs

L'exemple précédent est un programme Datalog (deux faits et une règle) non récursif.

Exemple

Un programme Datalog récursif :
$Ancêtre(x_1, x_2) \vdash Parent(x_1, x_2)$
$Ancêtre(x_1, x_3) \vdash Parent(x_1, x_2), Ancêtre(x_2, x_3)$

Datalog dans le cadre de ce cours

- Datalog non récursif
- Base de données $\Rightarrow$ un ensemble de faits
- Une règle $\Rightarrow$ une requête

Schéma d'un programme Datalog

Definition

(Schéma d'un programme Datalog)
Le symbole de base de données d'un programme Datalog P, noté $S(P)$, est défini par :
$S(P) = \{R \mid R$ est un symbole de relation qui apparaît dans un littéral d'une règle de $P\}$

Remarque

On infère la structure d'une base de données à partir d'un programme Datalog

Exemple (rappel)

Etudiants

Name	Age	Address	Dept
Vidal	21	Clermont	Computing
Dupond	35	Paris	Maths
Durand	45	Lyon	Linguistics
Dan	34	Lyon	Computing

Departements

Loc	Mgr	Dept
Bât A	Ullman	Computing
Bât B	Mendelson	Maths
Bât C	Dupond	Linguistics

Programme Datalog correspondant à l'exemple

Soit P_1 le programme suivant réduit à des faits :

$Etudiant(Vidal, 21, Clermont, Computing)$
$Etudiant(Dupond, 35, Paris, Maths)$
$Etudiant(Durand, 45, Lyon, Linguistics)$
$Etudiant(Dan, 34, Lyon, Computing)$
$Departement(BâtA, Ullman, Computing)$
$Departement(BâtB, Mendelson, Maths)$
$Departement(BâtC, Dupond, Linguistics)$

Exemple

$S(P_1) = \{Etudiant, Departement\}$

- *Les attributs ont disparu !*
- *Dans un prédicat $R(x_1, \ldots, x_n)$, chaque position i représente un attribut (qui n'a pas besoin d'être nommé)*
- *Plus de différence entre **schéma de relation et relation** pour un symbole de relation donné*

Sémantique d'un programme Datalog

Repose sur les notions suivantes :

- Substitution des variables d'un littéral
- Substitution des variables d'une règle
- Notion de vérité d'un littéral par rapport à une substitution et un programme Datalog
- Notion de vérité d'une règle par rapport à une substitution et un programme Datalog

Substitution de variables

Definition

(Substitution de variables)
Soit C une règle Datalog et $\{x_1, \ldots, x_q\}$ les variables apparaissant dans les littéraux du corps de C.
Une **substitution** θ pour C est l'ensemble des affectations $\{x_1/v_1, \ldots, x_q/v_q\}$, où les v_i sont des constantes.
Notons $C(\theta)$ la règle résultant de l'application de la substitution θ aux littéraux de C.

$\Rightarrow$ tous les littéraux de la règle $C(\theta)$ sont des *atomes clos*.

Vérité d'un littéral

$\Rightarrow$ se définit pour une substitution des variables de la règle contenant le littéral et une base de données

Un **littéral** L dans le corps d'une règle C d'un programme Datalog P est **vrai** (par rapport à une substitution θ pour C) si l'une des conditions suivantes est satisfaite :

- $\theta(L)$ est un atome clos atomique de la forme $R(v_1, \ldots, v_k) \in P$
- $\theta(L)$ est une égalité $v = v$ où v est une constante
- $\theta(L)$ est un atome clos de la forme $\neg R(v_1, \ldots, v_k)$ et $R(v_1, \ldots, v_k) \notin P$
- $\theta(L)$ est un littéral négatif de la forme $\neg(v_i = v_j)$, où v_i et v_j sont des constantes distinctes, i.e., $v_i \neq v_j$

Vérité d'une règle

$\Rightarrow$ se définit pour une substitution et un programme Datalog

Une **règle** C dans un programme P est **vraie** (par rapport à une substitution θ pour C) si chacun des littéraux du corps de C est vrai par rapport à θ et P

Sémantique d'un programme Datalog

La sémantique d'un programme Datalog P, notée $Meaning(P)$, est l'ensemble des faits (i.e. la base de données) résultant de l'augmentation de l'ensemble des faits initiaux de P par tous les nouveaux faits possibles de la forme $\theta(L)$, où

- θ est une substitution qui rend une règle C de P vraie et
- L est la tête de C.

Une requête sera représentée par une ou plusieurs règles Datalog

La sémantique d'un programme Datalog = Résultat de la requête

Algorithme Meaning

Algorithme Meaning(P)
Data: P, un programme Datalog
Result: *Result*, tous les faits possibles de P
$Result := \emptyset$;
$TMP := \{\langle\rangle\}$;
while $TMP \neq Result$ **do**
  $TMP := Result$;
  $Im := \emptyset$;
  foreach *rule* $C \in P$ *and substitutions* θ *for* C *such that* C *is true with respect to* θ *and Result* **do**
    $Im := Im \cup \{\theta(L)\}$ where L is the head of C
  $Result := Result \cup Im$
return($Result$) ;

Programmes peu sûrs

Une requête Datalog a un sens uniquement si la relation qui peut être dérivée en l'exécutant est finie

Exemple

$Result(x) \vdash Etudiant(x_1, x_2, x_3, x_4)$
$Result(x_1, x_2, x_4) \vdash \neg Etudiant(x_1, x_2, x_3, x_4)$

Programme Datalog sûrs

Peut on trouver des restrictions qui permettent de caractériser les programmes Datalog sûrs ?
Deux approches :

- une basée sur le statut des variables (similaire au calcul relationnel)
- une basée sur la notion de substitution sûre (pas faite ici)

Statut des variables

Une variable x apparaît *positivement* dans une règle C ssi :

- x apparaît dans un prédicat $R(y_1, \ldots, x, \ldots, y_k)$, qui est un littéral positif dans le corps de C, ou
- x apparaît dans une formule d'égalité $x = v$, qui est un littéral positif dans le corps de C, où v est une constante, ou
- x apparaît dans une formule d'égalité $x = y$ ou $y = x$, qui est un littéral positif dans le corps de C, où y est une variable qui apparaît positivement dans C

Programme Datalog sûrs

Definition

(Programme Datalog sûrs)
Une **règle** Datalog C est dite **sûre** si toutes les variables qui apparaissent dans les littéraux de C (dans le corps ou dans la tête) apparaissent positivement dans C
Un **programme** Datalog P est dit **sûr** si toutes les règles de P sont sûres

Remarques

- Dans une règle Datalog sûre C, toutes les variables qui apparaissent dans la tête doivent apparaître au moins une fois dans le corps de C.
- Toutes les variables apparaissant dans les littéraux négatifs du corps d'une règle sûre doivent également apparaître positivement dans au moins une formule atomique du corps de la règle

Exemple (rappel)

Etudiant(*Vidal*, 21, *Clermont*, *Computing*)
Etudiant(*Dupond*, 35, *Paris*, *Maths*)
Etudiant(*Durand*, 45, *Lyon*, *Linguistics*)
Etudiant(*Dan*, 34, *Lyon*, *Computing*)
Departement(*BâtA*, *Ullman*, *Computing*)
Departement(*BâtB*, *Mendelson*, *Maths*)
Departement(*BâtC*, *Dupond*, *Linguistics*)

Exemples de requêtes Datalog sur P_1

- "Donner la liste de tous les étudiants"
  $Result(x_1, x_2, x_3, x_4) \vdash Etudiant(x_1, x_2, x_3, x_4)$
- "Donner les noms et départements de tous les étudiants"
  $Result(x_1, x_4) \vdash Etudiant(x_1, x_2, x_3, x_4)$
- "Donner les noms et âges des étudiants qui sont soit inscrits dans le département *Linguistics* soit habitent à *Clermont*"
  $Result(x_1, x_2) \vdash Etudiant(x_1, x_2, x_3, x_4), x_4 = \text{`}Linguistics\text{`}$
  $Result(x_1, x_2) \vdash Etudiant(x_1, x_2, x_3, x_4), x_3 = \text{`}Clermont\text{`}$
- "Donner les noms des étudiants non parisiens qui ne sont pas inscrits dans le département *Computing*"
  $Result(x_1) \vdash Etudiant(x_1, x_2, x_3, x_4), \neg(x_3 = \text{`}Paris\text{`}), \neg(x_4 = \text{`}Computing\text{`})$

Exemples (cont.)

On suppose avoir des faits nommés $Etudiant1$ et $Etudiant2$

- Union entre $Etudiant1$ et $Etudiant2$
  $Result(x_1, x_2, x_3, x_4) \vdash Etudiant1(x_1, x_2, x_3, x_4)$
  $Result(x_1, x_2, x_3, x_4) \vdash Etudiant2(x_1, x_2, x_3, x_4)$
- Différence entre $Etudiant1$ et $Etudiant2$
  $Result(x_1, x_2, x_3, x_4) \vdash$
  $Etudiant1(x_1, x_2, x_3, x_4), \neg Etudiant2(x_1, x_2, x_3, x_4)$
- Intersection entre $Etudiant1$ et $Etudiant2$
  $Result(x_1, x_2, x_3, x_4) \vdash$
  $Etudiant1(x_1, x_2, x_3, x_4), Etudiant2(x_1, x_2, x_3, x_4)$
- Division entre $R_1(X, Y)$ et $R_2(Y)$:
  $Result(x_1) \vdash R_1(x_1, x_3), \neg DIFF(x_1)$
  $DIFF(x_1) \vdash PROD(x_1, x_2), \neg R_1(x_1, x_2)$
  $PROD(x_1, x_2) \vdash R_1(x_1, x_3), R_2(x_2)$

Equivalence des langages

Les trois langages vus sont équivalents : résultat fondamental des bases de données

$\Rightarrow$ Comment énoncer ce résultat ?

$\Rightarrow$ Esquisse de la preuve

Notions préliminaires

Deux requêtes Q_1 et Q_2 sur un schéma de BD R sont dites **équivalentes**, noté $\boldsymbol{Q_1 \equiv Q_2}$, si pour toute base de données $\boldsymbol{d}$ sur R, $\boldsymbol{Q_1(d) = Q_2(d)}$.
$\Rightarrow$ quelque soit la base de données, les réponses doivent être les mêmes

Un langage de requêtes $\boldsymbol{L_1}$ est dit **contenu** dans un langage de requêtes $\boldsymbol{L_2}$, noté $\boldsymbol{L_1 \preceq L_2}$, si pour toute requête $\boldsymbol{Q_1}$ de $\boldsymbol{L_1}$, il existe une requête $\boldsymbol{Q_2}$ de $\boldsymbol{L_2}$ tel que $\boldsymbol{Q_1 \equiv Q_2}$.

$\boldsymbol{L_1}$ est dit **équivalent** à $\boldsymbol{L_2}$, noté $\boldsymbol{L_1 \equiv L_2}$, si $\boldsymbol{L_1 \preceq L_2}$ et $\boldsymbol{L_2 \preceq L_1}$.

Théorême

Les trois langages relationnels suivants sont équivalents :

- L'algèbre relationnelle
- Le calcul relationnel du domaine autorisé
- $nr-datalog^{\neg}$ (Datalog sûr non récursif avec la négation)

Résultat fondamental des bases de données

Esquisse de la preuve

On peut montrer que :

1. l'algèbre est incluse dans le calcul autorisé
2. le calcul autorisé est inclu dans Datalog
3. Datalog est inclu dans l'algèbre

Considérons (1) : la preuve se fait par **induction** sur le nombre d'opérateurs algébriques d'une requête
L'hypothèse d'induction H(n) peut se formuler ainsi : "Soit Q une requête algébrique avec n opérateurs algébriques. On suppose qu'il existe une requête Q' du calcul autorisé tel que Q et Q' soit équivalente."
Preuve : Montrer H(0) et en supposant H(n), montrer H(n+1)

SQL : un rapide tour d'horizon

SQL vs Modèle relationnel

$\Rightarrow$ on se focalise sur l'implantation de l'algèbre relationnelle dans un langage pratique nommé SQL.

- Version nommée pour la structure du modèle relationnel
- Plus de distinction entre schéma de relation et relation : notion de table pour désigner les deux
- Une table SQL n'est pas un ensemble de tuples, mais un multi-ensemble de tuples, principalement pour des raisons d'efficacité (coût engendré par les tris pour supprimer les doublons) (voir diapo 79)

La sémantique de SQL

Une relation SQL n'est pas un ensemble de tuples ! ! !
$\Rightarrow$ mais un multi ensemble (ou bag)

Exemple

{ 1,2,2,3 } est un multi-set, i.e. plusieurs occurrences possibles de la même valeur, pas d'ordre.

Problème : les propriétés usuelles sur les ensembles ne sont plus toujours vérifiées sur les multi ensembles.

Exemple

$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$
Soit $R = S = T = \{1\}$, *on a alors*
$R \cap (S \cup T) = \{1\} \cap \{1, 1\} = \{1\}$ *et*
$(R \cap S) \cup (R \cap T) = \{1\} \cup \{1\} = \{1, 1\}$

SQL (Structured Query Language)

SQL : Langage d'interrogation des données extrêmement populaire, utilisé partout ...

Langage concret construit sur les trois langages formels étudiés + du sucre syntaxique

Des différences notoires, dont le fait qu'une table est un multi-ensemble de tuples et non plus un ensemble

Normalisé par l'ANSI en 1992 (SQL-92) puis en 1999 (SQL-99), différents niveaux de compatibilité
$\Rightarrow$ Chaque SGBD a son propre SQL, typiquement pour tous les aspects hors-normes : built-in function, fermeture transitive
$\Rightarrow$ Ne faire confiance qu'à la documentation du SGBD utilisé plus précisément, à la grammaire SQL de la version du SGBD utilisé (par ex doc SQL d'Oracle 10.2)

Syntaxe d'une requête SQL

De base, une requête SQL comporte trois clauses :

```
SELECT <liste d'attributs>
FROM     <listes relations>
[ WHERE <conditions sur les lignes> ]
```

$\Rightarrow$ requête SFW
La clause SELECT permet de coder la *projection*
La clause WHERE permet de coder la *selection*. Elle est optionnelle.
Le résultat est une relation sur le schéma défini par la clause SELECT.
Une requête SQL peut être nommée $\Rightarrow$ on parle de **vue**.

Passerelles entre ensembles et multi ensembles

Possible grâce aux mots clés DISTINCT et ALL

- multi ensemble vers ensemble : DISTINCT
- ensemble vers multi ensemble : ALL

Les requêtes SFW sont **par défaut** définies sur des multi-ensembles
$\Rightarrow$ pour forcer la sémantique ensembliste, utiliser le mot clé 'DISTINCT' après 'SELECT'
Les opérateurs ensemblistes (UNION, INTERSECT, MINUS) force par défaut la sémantique ensembliste ! ! !
$\Rightarrow$ pour forcer la sémantique multi ensemble, utiliser le mot clé 'ALL' après 'UNION', ...
$\Rightarrow$ Etre prudent sur les conversions, et se méfier du coût (opération de TRI) engendré par DISTINCT

Rappel : base de données

Personnes	nss	nom	prenom	age
	12	Aymard	Serge	45
	45	Fenouil	Solange	35

Departements	dep	adresse
	Math	Carnot
	Info	Cézeaux

Travaillent	nss	dep	activite
	12	Math	Prof
	45	Math	MdC
	45	Info	MdC

Requête SFW

Exemple

Q : "Nom et prénom des personnes de 30 ans ?"

- $\pi_{nom,prenom}(\sigma_{age=30}(Personnes))$
- $ans(Q_c, d)$ *avec* $Q_c = \{x_2 : nom, x_3 : prenom, x_4 : age | \exists x_1 : nss(Personne(x_1, x_2, x_3, x_4) \wedge x_4 = 30)\}$
- $Q(x_1, x_3) \vdash Personne(x_1, x_2, x_3, 30)$

$\Rightarrow$ *en SQL :*

```
SELECT DISTINCT nom, prenom
FROM   Personnes
WHERE  age = 30;
```

Les mots réservés du langage ne sont pas sensibles à la casse

Le renommage en SQL

Possible sur les attributs dans la clause 'SELECT' et sur les relations dans la clause 'FROM'
Mot clé réservé `AS`

```
SELECT DISTINCT nom [AS] "Mes enfants"
FROM     Personnes [AS] P
WHERE  [P.]age = 12;
```

AS est optionnel !
Résolution d'ambiguïtés :
On prefixe l'attribut par le nom de sa relation, i.e. `relation.nom`

Conditions complexes

Mots clés `AND`, `OR`, `NOT` et parenthèses pour former des expressions complexes
Permettent de former des formules de sélection complexes

```
SELECT DISTINCT nom, prenom
FROM   Personnes
WHERE  (age = 12) AND (nss > 8932);
```

1° douceurs syntaxiques : les caractères jockers

$\Rightarrow$ ne s'exprime pas en algèbre relationnelle

Dans la clause 'SELECT' :
On peut lister tous les attributs avec le caractère '*'
Dans la clause 'WHERE' :

- Mot clé `LIKE` avec les caractères jockers : %, _

```
SELECT DISTINCT nom
FROM     Personnes
WHERE  prenom LIKE 'J%';
```

- Expressions arithmétiques usuelles sur les valeurs retournées

```
SELECT DISTINCT nom, age-10
FROM     Personnes
```

Jointure naturelle

Exemple

"Nom, prénom des personnes et les départements dans lesquels ils travaillent ?"

- $\pi_{nom,prenom,dep}(Personnes \bowtie Travaillent)$
- $\{x_2 : nom, x_3 : prenom, y_1 : dep | \exists x_1 : nss(\exists x_4 : age(Personne(x_1, x_2, x_3, x_4))) \wedge \exists y_1 : nss(\exists y_2 : dep \exists y_3 : activite(Travaille(y_1, y_2, y_3)))) \wedge x_1 = y_1\}$
- $Q(x, y, z) \vdash Personne(x', x, y, \_), Travaille(x', \_, z)$

$\Rightarrow$ *en SQL :*

```
SELECT DISTINCT nom, prenom, dep
FROM Personnes NATURAL JOIN Travaillent
```

```
SELECT DISTINCT nom, prenom, dep
FROM   Personnes, Travaillent
WHERE  Personnes.nss = Travaillent.nss;
```

Que pensez vous de l'hypothèse URSA pour les noms d'attributs en SQL ?

Produit cartésien

Il suffit de ne pas expliciter d'attributs de jointure

```
SELECT DISTINCT *
FROM   Personnes, Travaillent
```

$\Rightarrow$ Si un même attribut A apparaît dans les 2 tables, un indice (1 pour le 1er opérande et 2 pour le second) est ajouté à A, i.e. $A1$ et $A2$

Les opérateurs ensemblistes en SQL

L'union, l'intersection et la différence sont représentés respectivement par les mots clés UNION, INTERSECT et MINUS.
Les deux opérandes doivent avoir le même schéma.
Sur l'exemple, schema(R) = schema(S) = { A, B }.

```
SELECT A, B
FROM   R
UNION
SELECT A, B
FROM S
```

Ces trois opérations forcent la sémantique ensembliste par défaut

2° douceurs syntaxiques : composition de requêtes

Le résultat d'une requête SFW peut être utilisé dans la clause FROM ou WHERE d'une autre requête
$\Rightarrow$ En cas d'ambiguité, utiliser des parenthèses

Les sous-requêtes de la clause 'WHERE' sont introduites par les mots clés `IN`, `EXISTS`, `ANY`, `ALL`

Exercice
Ecrire en SQL l'expression algébrique de la division avec $-, \pi, \times$ *(cf. diapo 14)*

Opérateur IN

Opérateur IN :

```
SELECT ...
FROM  R
WHERE X IN (SELECT Y
            FROM   S
            WHERE  ...)
```

Soit $t \in R$. t est sélectionnée si $\exists t' \in S$ tq $t[X] = t'[Y]$

```
SELECT distinct nom, prenom
FROM   Personnes
WHERE  nss IN (SELECT  distinct nss
               FROM    Travaillent)
```

Possibilité d'utiliser NOT IN

Opérateur EXISTS

```
SELECT ...
FROM   R
WHERE EXISTS (SELECT  *
              FROM    S
              WHERE   C)
```

La condition C doit comporter au moins une condition impliquant un attribut de R
Soit $t \in R$. t est sélectionnée si $\exists t' \in S$ tq t et t' satisfont C

```
SELECT DISTINCT nom, prénom
FROM   Personnes P
WHERE EXISTS (SELECT  *
              FROM    Travaillent T
              WHERE   P.nss = T.nss)
```

Le quantificateur existentiel ANY

$\Rightarrow$ généralise IN
Permet d'exprimer les requêtes du type "il existe ..."

```
SELECT ...
FROM  R
WHERE X bop ANY (SELECT DISTINCT Y
                 FROM   S
                 WHERE ...)
```

'bop' est un opérateur binaire de comparaison classique, e.g. $=, <, >, \ldots$ qui retourne vrai ou faux
Soit $t \in R$. t est sélectionnée si $\exists t' \in S$ tq $t[X]$ **bop** $t'[Y]$

Remarque : IN $\equiv =$ ANY

Le quantificateur universel ALL

Permet d'exprimer les requêtes du type "pour tous les ..."

```
SELECT ...
FROM  R
WHERE X bop ALL (SELECT DISTINCT Y
                 FROM   S
                 WHERE ...)
```

'bop' est un opérateur binaire de comparaison classique, e.g. $=, <, >, \ldots$ qui retourne vrai ou faux
Soit $t \in R$. t est sélectionnée si $\forall t' \in S$ tq $t[X]$ **bop** $t'[Y]$

Exercice

Exprimer les deux expressions suivantes avec EXISTS et NOT EXISTS respectivement

```
SELECT ...
FROM  R
WHERE X bop ANY (SELECT DISTINCT Y FROM S WHERE C);
```

```
SELECT ...
FROM  R
WHERE X bop ALL (SELECT DISTINCT Y FROM S WHERE C);
```

3° douceurs syntaxiques : fonctions d'agrégation

Les fonctions d'agrégation permettent de répondre aux requêtes du type :
"Quel est le nombre d'étudiants inscrits dans chaque département"
"Quel est l'âge moyen des étudiants"
Possibilités de calculer des fonctions sur une colonne du résultat
$\Rightarrow$ la requête retourne une colonne et une ligne ! ! !
Fonctions arithmétiques classiques : SUM, AVG, MIN, AVG, COUNT ... (dépend du SGBD utilisé)

```
SELECT count(*) "Nombre de lignes"
FROM Personnes;

SELECT AVG(age)
FROM Personnes;
```

Regroupement des données : la clause GROUP BY

Permet de généraliser les fonctions précédentes : le calcul des agrégations est possible sur des éléments d'une partition
Permet d'utiliser les fonctions d'agrégation sur des sous ensembles de valeurs.
Syntaxe :

```
SELECT <liste d'attributs>
FROM     <listes relations>
WHERE <conditions sur les lignes>
GROUP BY <liste d'attributs>
HAVING <conditions sur les éléments de la partit
```

La partition de la table résultat se fait sur les attributs spécifiés après le GROUP BY.
Le résultat de la requête comporte autant de lignes que d'éléments dans la partition.

Suite

Exemple

Achat(id, client, date, prix)
"Prix moyen des achats effectués par chaque client"

```
SELECT avg(prix), client
FROM  Achat
GROUP BY client
```

L'utilisation de la clause GROUP BY impose des restrictions sur la liste des attributs de la clause SELECT :

- soit un attribut agrégé, e.g. avec SUM, AVG ...
- soit un attribut qui apparaît déjà dans la clause GROUP BY

Conditions de sélection sur les regroupements

La clause WHERE permettait de définir des sélections sur les lignes du résultat d'une requête
$\Rightarrow$ la clause HAVING va permettre de définir des sélections sur les éléments d'une partition

Exemple

Achat(id, client, date, prix)
"Prix moyen des achats effectués par chaque client en quantité au moins egàle à 2"

```
SELECT avg(prix), client
FROM     Achat
GROUP BY client
HAVING count(*)>=2
```

4° douceurs syntaxiques : Les valeurs nulles

Très nombreuses en pratique
Rend la conception de requêtes parfois subtile ...
Sémantique des valeurs nulles : une condition logique avec un NULL retourne toujours FAUX
Mot clé IS [NOT] NULL
Expression de sélection :

```
SELECT A, B, C
FROM R
WHERE A IS NOT NULL OR (B IS NULL AND C IS NULL);
```

Jointures externes

Nécessaires lorsque des valeurs nulles existent sur des attributs de jointures

Exemple

R(A, B, C) – des nulles existent sur C
S(C, D, E) – pas de nulles sur C

```
SELECT A, B, D
FROM R, S
WHERE R.C (+) = S.C
```

$\Rightarrow$ Jointure externe à gauche ou à droite

D'autres syntaxes existent ... pour exprimer exactement la même requête

Morale de l'histoire : **Le sucre, on a parfois tendance à trop en mettre !**

Outline

Les contraintes d'intégrité

L'idée de base est de fournir des moyens qui permettent de restreindre les données "valides" d'une base de données
$\Rightarrow$ on ne peut pas insérer tout et n'importe quoi
$\Rightarrow$ La mise en œuvre de cette notion se fait via des **contraintes d'intégrité** : " déclarations logiques qui permettent de restreindre le domaine actif d'une base de données"

Exemple

Un numéro de sécurité social ne peut pas être le même pour deux personnes différentes.
$\Rightarrow$ *notion de clé*
Un employé ne peut pas être affecté à un numéro de projet qui n'existe pas
$\Rightarrow$ *notion de clé étrangère*

Dépendances fonctionnelles (DF)

Concept très important, inhérent à la **modélisation des données** $\Rightarrow$ à la base de la *théorie de la conception des bases de données*.
Intérêts pratiques :

- permet de définir formellement la notion de **bons schémas** $\Rightarrow$ ceux pour lesquels il n'existe pas d'anomalies lors d'insertions, de modifications, ou de suppressions de tuples.
- généralise la notion de **clé** (dans les SGBD, clé primaire et contrainte d'unicité).

Exemple

Exemple

Soit $\mathcal{U} = \{id, nom, adresse, cnum, desc, note\}$ *un univers décrivant des étudiants et des cours.*
Soient les deux schémas de BD suivants :

- $R_1 = \{Donnees\}$ *avec* $schema(Donnees) = \mathcal{U}$
- *et* $R_2 = \{Etudiant, Cours, Affectation\}$ *avec* $schema(Etudiant) = \{id, nom, adresse\}$, $schema(Cours) = \{cnum, desc\}$, $schema(Affectation) = \{id, cnum, note\}$

Comment peut-on évaluer ces deux schémas de données ?
Lequel est "meilleur" ? Pourquoi ? Selon quels critères ?

NB : données de $R_1 \simeq$ données d'un tableur

Un exemple de BD sur R_1

Données	id	nom	adresse	cnum	desc	note
	124	Jean	Paris	F234	Philo I	A
	456	Emma	Lyon	F234	Philo I	B
	789	Paul	Marseille	M321	Analyse I	C
	124	Jean	Paris	M321	Analyse I	A
	789	Paul	Marseille	CS24	BD I	B

Quels sont les problèmes ?

- l'information est *redondante*.
  Ex : '124, Jean, Paris' apparaît dans deux lignes différentes
  $\Rightarrow$ **Anomalie de modification**
  Une modification sur une ligne peut nécessiter des modifications sur d'autres lignes

Exemple (suite)

- *Certaines informations dépendent de l'existance d'autres informations*
  Ex : Le cours 'CS24' de BD dépend de l'existance de Paul.
  $\Rightarrow$ **Anomalie de suppression**

- *Valeurs manquantes* (pas de valeurs nulles dans le cours)
  Ex : Soit '145, Calixte, Aubenas' un nouvel étudiant. On ne peut l'insérer que si l'on connait un de ses cours et sa note dans ce cours, à moins de permettre les valeurs nulles.
  $\Rightarrow$ **Anomalie d'insertion**

$\Rightarrow$ Le moyen simple qui permet d'éviter ces problèmes est l'étude de contraintes particulières, les *dépendances fonctionnelles*.

Syntaxe et sémantique des DF

- Définition (ou syntaxe)
  Une DF sur un schéma R est une déclaration de la forme :
  $R : X \rightarrow Y$, où $X, Y \subseteq \mathit{schema}(R)$
- Satisfaction d'une DF (ou sémantique)
  Une DF $R : X \rightarrow Y$ est *satisfaite* par une relation r sur R, noté $r \models X \rightarrow Y$, ssi $\forall t_1, t_2 \in r$ si $t_1[X] = t_2[X]$ alors $t_1[Y] = t_2[Y]$

Exemple

Que pensez-vous de ces deux assertions ?
$\mathit{Donnees} \models \mathit{id} \rightarrow \mathit{nom}, \mathit{adresse}$
$\mathit{Donnees} \not\models \mathit{id} \rightarrow \mathit{note}$

Raisonnement sur les DF

On peut raisonner sur les DF, par exemple $A \rightarrow B, C$ est équivalent à $A \rightarrow B, A \rightarrow C$

Définition informelle d'une **clé** : une clé est un ensemble (minimal) d'attributs tel que il n'est pas possible d'avoir deux tuples avec la même valeur sur la clé (sinon serait un contre-exemple).

$\Rightarrow$ Notions théoriques sous-jacentes développées après

Les formes normales (suite de la 1FN)

Permettent de spécifier formellement la notion intuitive de bons schémas
Pour les DFs, plusieurs formes normales (FN) existent : 1FN (moins restrictive) , 2FN, 3FN, FN de Boyce-Codd (FNBC) (plus restrictive).

Idée de la FNBC : *ne représenter l'information qu'une seule fois*.

Definition

Un symbole de relation R est en FN de Boyce-Codd par rapport à un ensemble F de DF ssi pour toutes les dépendances (non triviales) de F, leurs parties gauches sont des clés.

$\Rightarrow$ évite les problèmes de redondance et d'anomalie identifiés précédemment

Retour sur l'exemple

Exercice

Peupler une base de données définie sur R_2 à partir des données de R_1.

Qu'en est il des problèmes de MAJ soulevés avec R_1 ?

Etant donné $\mathcal{U}$ et les dépendances fonctionnelles suivantes : $F = \{\{id\} \rightarrow \{nom, adresse\}, \{cnum\} \rightarrow \{desc\}, \{cnum, id\} \rightarrow \{note\}\}$

Que pensez vous des parties gauches des DF sur R_1 ?
Sur R_2 ?

Dépendances d'inclusion (DI)

Concept clé pour la conception de schémas de BDs
$\Rightarrow$ comparable aux DFs, i.e. permet de dissocier les bons des mauvais schémas

Les DF permettent de spécifier des contraintes intra-relations
$\Rightarrow$ les DIs permettent de spécifier des contraintes inters-relations

Les DF généralisent les clés
$\Rightarrow$ les DIs généralisent les **clés étrangères**

Retour sur la BD sur R_2

En terme de mises à jour de la BD, on peut encore insérer des données non valides.

Exemple

SANS violer aucune DF, on peut donner une note à un étudiant qui n'existe pas et ce, dans un cours qui n'existe pas non plus ! ! !

$\Rightarrow$ l'idée est à nouveau de restreindre les insertions permises en spécifiants deux nouvelles contraintes : " une note ne peut être saisie que pour un étudiant qui existe dans la relation Etudiants et un cours qui existe dans la relation Cours"

Syntaxe des DI

ATTENTION : on parle de *séquences d'attributs distincts* pour les DI, à ne pas confondre avec les ensembles.
Stricto sensus, on devrait redéfinir les opérateurs ensemblistes usuels pour les séquences ...

Definition

Soient **R** un schéma de base de données R_1 et R_2 deux schémas de relation de **R**.
Une dépendance d'inclusion sur **R** est une *déclaration* de la forme $R_1[X] \subseteq R_2[Y]$, avec X et Y des séquences d'attributs distincts appartenant à $schema(R_1)$ et $schema(R_2)$ respectivement.

Exemple

$Affectation[id] \subseteq Etudiant[id]$
$Affectation[cnum] \subseteq Cours[cnum]$

Sémantique des DI

Definition

Soient d une base de données sur $\mathbf{R}$ et $r_1, r_2 \in d$ définies sur $R_1, R_2 \in \mathbf{R}$ respectivement.
Une dépendance d'inclusion $R_1[X] \subseteq R_2[Y]$ est *satisfaite* par d, noté $d \models R_1[X] \subseteq R_2[Y]$, ssi $\forall t_1 \in r_1$, $\exists t_2 \in r_2$ tq $t_1[X] = t_2[Y]$

NB : $d \models R_1[A_1, \ldots, A_n] \subseteq R_2[B_1, \ldots B_n]$, ssi
$\pi_X(r_1) \subseteq \rho_{B_1 \to A_1}(\ldots(\rho_{B_n \to A_n}(\pi_Y(r_2)))\ldots)$

Exemple

{ Affectations, Etudiants } $\models$ *Affectation[id]* $\subseteq$ *Etudiant[id]*
{ Affectations, Cours } $\models$ *Affectation[cnum]* $\subseteq$ *Cours[cnum]*

$\Rightarrow$ Les clés étrangères sont des cas particuliers de *dépendances d'inclusion*

Exercice

Donner une définition des clès étrangères en fonction des clés et des DIs.

Règles d'inférence pour les dépendances d'inclusion

Soit I un ensemble de DI sur un schéma de base de données $\mathbf{R} = \{R_1, \ldots, R_n\}$
Système d'axiomes de Casanova et al. :

- Reflexivité : si $X \subseteq \mathit{schema}(R)$, alors $I \vdash R[X] \subseteq R[X]$
- Projection et permutation : si $I \vdash R_1[X] \subseteq R_2[Y]$ où $X = \{A_1, \ldots, A_m\} \subseteq \mathit{schema}(R_1)$, $Y = \{B_1, \ldots, B_m\} \subseteq \mathit{schema}(R_2)$ et $i_1, \ldots, i_k$ est une séquence de nombres naturels de $\{1, \ldots, m\}$, alors $I \vdash R_1[A_{i_1}, \ldots, A_{i_k}] \subseteq R_2[B_{i_1}, \ldots, B_{i_k}]$
- Transitivité : si $I \vdash R_1[X] \subseteq R_2[Y]$ et $I \vdash R_2[Y] \subseteq R_3[Z]$, alors $I \vdash R_1[X] \subseteq R_3[Z]$

Théorême

Le système d'axiomes de Casanova et al. est juste et complet

Retour sur les dépendances fonctionnelles

Raisonnement sur les DF

Les DF sont à la source de la théorie de la conception des bases de données

Se retrouvent sous diverses formes :

- Notion d'implication en Analyse Formelle de Concepts
- Notion de règles d'association en fouille de données

$\Rightarrow$ **Pré-requis pour aller plus loin**

Ensemble de DF

Soient F et G deux ensembles de DF sur R.
$\Rightarrow F$ et G peuvent être différents syntaxiquement mais *équivalents* sémantiquement, i.e. ils représentent les mêmes dépendances

Le problème de base est connu sous le terme du *problème d'implication* : "Etant donnés un ensemble F de DF sur R et une DF $X \rightarrow Y$ sur R, décider si F implique $X \rightarrow Y$"
Autrement dit, peut on déduire ou prouver $X \rightarrow Y$ à partir de F ?

Deux approches classiques :

- Théorie des modèles
- Théorie de la preuve

Théorie des modèles

Pour résoudre le problème d'implication, on examine toutes les bases de données qui vérifient F et on teste si elles vérifient aussi $X \rightarrow Y$.

Plus formellement :
Soient F un ensemble de DF sur R et $X \rightarrow Y$ une DF sur R.
F implique $X \rightarrow Y$, notée $F \models X \rightarrow Y$, ssi pour toute relation r sur R, si $r \models F$, alors $r \models X \rightarrow Y$

$\Rightarrow$ on parle d'*implication logique*

Théorie de la preuve

Pour résoudre le problème d'implication, on construit une preuve de $X \to Y$ à partir de F en appliquant des règles d'inférence.

- Règles d'inférence d'Armstrong
  1. Réflexivité : si $Y \subseteq X \subseteq \mathit{schema}(R)$, alors $F \vdash X \to Y$
  2. Augmentation : si $F \vdash X \to Y$ et $W \subseteq \mathit{schema}(R)$, alors $F \vdash XW \to YW$
  3. Transitivité : si $F \vdash X \to Y$ et $F \vdash Y \to Z$, alors $F \vdash X \to Z$
- Notion de preuve :
  Soient F un ensemble de DF sur R et $X \to Y$ une DF sur R.
  F implique $X \to Y$, notée $F \vdash X \to Y$, ssi on peut écrire une dérivation (ie. une preuve) qui à partir de F, utilise les règles d'inférence d'Armstrong pour aboutir en un nombre fini d'étape à $X \to Y$.

Equivalence des deux approches

Les deux approches **coïncident**, i.e. il existe une équivalence entre les deux approches

Théorême

Le système d'inférence d'Armstrong est juste ($F \vdash X \to Y \Rightarrow F \models X \to Y$) *et complet* ($F \models X \to Y \Rightarrow F \vdash X \to Y$).

cf Livres de la diapo 5 pour la preuve

En pratique, on pourra utiliser l'un ou l'autre des sigles ($\models$ ou $\vdash$) pour représenter l'implication.

Petite distraction ...

Une "ola" dans un amphi

On va simplifier la représentation de l'amphi : On le voit comme une matrice avec n lignes (les rangées) et m colonnes (demande un peu plus d'imagination ...)
On considère l'univers comme l'ensemble des positions possibles, en supposant que toutes les positions sont occupées par un étudiant

$\mathcal{U} = \{(i,j)|i = 1..n, j = 1..m\}$
$\Rightarrow \mathcal{U}$ représente donc l'ensemble des étudiants

$(1,1)$	$\ldots$	$(1,m)$
$\vdots$	$\vdots$	$\vdots$
$(n,1)$	$\ldots$	(n,m)

La colonne 1 est considérée "la plus à gauche" et la ligne 1 "la plus haute".

Règles du jeu

On considère les règles (ou **implications**) suivantes sur les étudiants : (que vous pouvez voir "comme des DFs" sur $\mathcal{U}$) :

- Si les étudiants d'une "colonne" de l'amphi vérifient une condition, alors le premier étudiant en haut de la colonne suivante (i.e. à droite) la vérifie aussi"
- Si un étudiant et son voisin situé la rangée immédiatement au dessus et à droite vérifient une condition, alors son voisin immédiatement à sa droite la vérifie aussi.

On peut modéliser ces règles comme suit :

- $F = \{\{(1,j),(2,j),\ldots,(n,j)\} \to \{(1,j+1)\} \mid j = 1..m-1\}$
- $F = F \cup$
  $\{\{(i,j),(i-1,j+1)\} \to \{(i,j+1)\} \mid i = 2..n, j = 1..m-1\}$

De façon plus concrète

Soit $E \subseteq \mathcal{U}$ un sous-ensemble des étudiants.

Supposons deux choses :

- les étudiants du groupe E décident une action, par exemple de lever les bras au ciel. C'est la condition à vérifier.
- Les règles de F permettent de "**propager**" cette action

Un regard différent sur les DFs

Quelques questions :

- Soit un étudiant dans l'amphi. S'il est le seul à lever les bras au ciel, que se passe t il ?
- Soit E les étudiants les plus à gauche (i.e. de la première colonne) de l'amphi.
  - S'ils lèvent les bras au ciel, que se passe t il ?
  - Si tous sauf un le font, que se passe t il ?
- Quels sont les plus petits groupes d'étudiants qui garantissent qu'il y ait une ola dans l'amphi ?

Les notions sur les DFs ont un caractère universel !

Fin de la distraction !

Algorithmique pour le problème d'implication

Ramener le problème d'implication $F \models X \rightarrow Y$ à un test de fermeture sur un ensemble d'attributs

Definition

Soit $X \subseteq schema(R)$ et F un ensemble de DF sur R
La fermeture de X par rapport à F, notée X_F^+, est définie par :

$$X_F^+ = \{A \in schema(R) \mid F \models X \rightarrow A\}$$

Propriété

$F \models X \rightarrow Y$ *ssi* $Y \subseteq X_F^+$

Algorithme ?

Algorithme de calcul de fermeture

Algorithme CLOSURE(X, F)
Data: X, un ens d'attribut ; F un ens de DF sur R
Result: X^+, la fermeture de X par rapport à F
$X^+ := X$;
fini := *false* ;
while *not fini* **do**
    fini := *true* ;
    foreach $W \rightarrow Z \in F$ **do**
        if $W \subseteq X^+$ *and* $Z \not\subseteq X^+$ **then**
            $X^+ := X^+ \cup Z$;
            fini := *false* ;
Return(X^+)

Fermé et système de fermeture

Soit F un ensemble de DFs sur R et $X \subseteq schema(R)$.

Definition

X est *fermé* par rapport à F si $X = X_F^+$

On note $CL(F)$ l'ensemble des fermés de F, i.e.
$CL(F) = \{X | X = X_F^+\}$
et $IRR(F)$ ses éléments irréductibles par intersection, i.e.
$IRR(F) = \{X \in CL(F) |$ pour tout $Y, Z \in CL(F)$
$(X = Y \cap Z) \Rightarrow (X = Y$ ou $X = Z)\}$

$\Rightarrow schema(R) \in CL(F)$
$\Rightarrow schema(R) \notin IRR(F)$ car $schema(R) = \cap\{\}$ par convention

Remarque

L'application $.^+_F$ *est un opérateur de fermeture sur* $schema(R)$

Equivalence des ensembles de DFs

Soient F et G deux ensembles de DF sur R.

Definition

La fermeture de F, notée F^+, est définie par
$F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$

Definition

F et G sont *équivalents*, noté $F \equiv G$, si $F^+ = G^+$

Propriété

$F \equiv G$ *ssi* $CL(F) = CL(G)$

Couverture d'ensemble de DF

Une *couverture* F (ou *base*) d'un ensemble K de DF est un ensemble de DF équivalent à K. Une couverture F de K est dite :

- *non redondante* si $\nexists G \subset F$ tel que $G^+ = F^+$
- *minimale* si $\nexists G$ tel que $G^+ = F^+$ et $|G| < |F|$
- *optimale* si $\nexists G$ tel que $G^+ = F^+$ et $||G|| < ||F||$ avec $||F|| = \sum_{X \to Y \in F} |X| + |Y|$

Retour sur la notion de clé

$\Rightarrow$ Les clés sont des cas particuliers de *dépendances fonctionnelles*

Soit F un ensemble de DF sur R et $X \subseteq \mathit{schema}(R)$.

- Superclé
  X est une *superclé* de R par rapport à F si
  $F \models X \rightarrow \mathit{schema}(R)$ (ou $X_F^+ = \mathit{schema}(R)$)
- Clé
  X est une *clé* (ou clé minimale) pour F ssi X est une superclé de R et $\nexists Y \subset X, Y$ superclé de R

Projection de F sur un sous ensemble d'attributs

Soient R, S deux symboles de relation tel que
$schema(S) \subset schema(R)$ et F un ensemble de DF sur R.

La projection de F sur S, notée $F[S]$, est définie par :
$F[S] = \{X \rightarrow Y \mid F \models X \rightarrow Y, X, Y \subseteq schema(S)\}$

Exemple

Soient $schema(R) = ABC, F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$ *et*
$F' = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$
On considère la décomposition suivante : $R_1 = AB, R_2 = BC$

Calculer $F[R_1], F[R_2], F'[R_1], F'[R_2]$

Forme normale

On suppose que F ne contient aucune DF triviale ($X \to Y$ avec $Y \subseteq X$)

Definition

Un attribut $A \in \mathit{schema}(R)$ est dit un *attribut prime* par rapport à F si A est inclu dans une clé de R par rapport à F.

Definition

Un symbole de relation R est en *3e forme normale* (3NF) par rapport à F si pour toute DF $X \to Y$ de F, X est une clé (pas forcément minimale) de F ou Y possède un attribut prime.

Definition

Un symbole de relation R est en *forme normale de Boyce-Codd* (BCNF) par rapport à F si pour toute DF $X \to Y$ de F, X est une clé (pas forcément minimale) de F.

Utilité et intérêt

Représentation "par l'exemple" d'ensemble de contraintes : on manipule des valeurs, visualisation plus simple des éventuels conflits, incohérences, mauvaise conception ...
Deux principaux domaines d'application :

- Conception par l'exemple
- Échantillonnage, compréhension des bases de données existantes

Idée : Représenter sur un exemple un ensemble de contraintes et que celui la, i.e. toutes les autres contraintes sont fausses

Relations d'Armstrong pour les DFs

Soit r une relation sur R, F un ensemble de DFs sur R et $X \to Y$ une DF sur R.

Definition

r est une *relation d'Armstrong* par rapport à F si $r \models X \to Y$ **si et seulement si** $F \models X \to Y$

Relations d'Armstrong pour les DFs (cont.)

Problème d'existence ?

- Existent toujours pour les DF standards (partie gauche des DFs non réduite à l'ensemble vide)

Caractérisation des relations d'Armstrong ?
On définit :

- $ag(t_1, t_2) = \bigcup\{A \in schema(R) | t_1[A] = t_2[A]\}$
- $ag(r) = \{ag(t_1, t_2) | t_1, t_2 \in r\}$

Soit r une relation sur R et F un ensemble de DFs standards sur R

Theorem

r est une relation d'Armstrong pour F ssi $IRR(F) \subseteq ag(r) \subseteq CL(F)$

Rappel : $X \in IRR(F)$ ssi pour tout $Y, Z \in CL(F)$
$(X = Y \cap Z) \Rightarrow (X = Y$ ou $X = Z)$

Construction

Principe de la construction : insérer autant de contre-exemples qu'il existe de DF non satisfaites !

Quel lien avec les ensembles en accord ?
$\Rightarrow ag(t_1, t_2)$ donne tous les contre-exemples des 2 tuples

Principe de la construction de relations d'Armstrong :

1. à partir de F, calculer une représentation de $CL(F)$
2. initialiser la relation d'Armstrong r avec un unique tuple, soit t_0 composé de 0
3. *(étape i)* pour chaque élément X de la représentation de $CL(F)$, ajouter un tuple t_i à r tq
   - $t_i[A] = 0$ pour tout $A \in X$ et
   - $t_i[B] = i$ pour tout $B \in (schema(R) - X)$

Outline

Avant propos

Deux approches principales :

- approche dite de la *relation universelle*
- approche dite conceptuelle

Relation universelle :
on spécifie des attributs (l'univers) puis des contraintes (principalement des DF) et on produit des schémas de relations dans une certaine forme normale

- 3FN : Algorithme de *synthèse*
- FNBC : Algorithme de *décomposition*

Approche conceptuelle :
on définit de nouvelles abstractions pour modéliser les données afin de se rapprocher "le plus possible" du monde réel que l'on souhaite modéliser.

Outline

Algorithmes de normalisation

Pré-requis : phase de modélisation du monde réel

- énumérer les attributs importants, i.e. construction de l'univers du discours $\mathcal{U}$
- spécifier les contraintes, principalement les dépendances fonctionnelles entre ces attributs

Utiliser les contraintes pour **décomposer** $\mathcal{U}$ en un ensemble de symboles de relation en FNBC

$\Rightarrow$ Assurer des **propriétés** pour cette décomposition :

$\Rightarrow$ `Algorithmes de normalisation` :

- Algorithme de décomposition
- Algorithme de synthèse

Quelles propriétés ?

Quelles sont les propriétés naturelles qu'il faudrait respecter ?

- Décomposition sans perte de jointure
  $\Rightarrow$ aucune donnée n'est perdue
- Préservation des dépendances
  $\Rightarrow$ toutes les dépendances sont conservées

Propriétés pas toujours atteignables

Définitions

Une *décomposition* d'un symbole de relation R est un ensemble de symboles de relations $\{R_1, \ldots, R_n\}$ tel que :

- $\forall R_i$, $schema(R_i) \subseteq schema(R)$, et
- $schema(R) = \bigcup_{i=1}^{n} schema(R_i)$

Définitions (suite)

Soient R un symbole de relation, F un ensemble de DF sur R et $\mathbf{R} = \{R_1, \ldots, R_n\}$ une décomposition de R.
R est une **décomposition sans perte (de jointure)** de $schema(R)$ par rapport à F si pour toute relation r sur $schema(R)$, on a :

$$r = \pi_{R_1}(r) \bowtie \ldots \bowtie \pi_{R_n}(r)$$

Définitions (fin)

R est une décomposition de $schema(R)$ qui **préserve les DF** de F si :

$$F^+ = (F[R_1] \cup \ldots \cup F[R_n])^+$$

Exemple

Soient $schema(R) = ABC$ *et* $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
On considère la décomposition en deux schémas $\{A, B\}, \{B, C\}$
Cette décomposition préserve les DF de F ?

Le problème de décomposition

Etant donnés

- un symbole de relation **R** et
- un ensemble de DFs F sur **R**

Trouver une décomposition de **R** qui vérifie *les* propriétés précédentes

Algorithme de synthèse

Principe :

- Déterminer une couverture minimale G de F,
- Réduire les parties gauches et droites des DF de G,
- Créer un symbole de relation pour chaque DF de G

Intuition : on réduit F de telle façon que toute DF de G donnera un symbole de relation
$\Rightarrow$ préserve les dépendances !

Calcul d'une couverture minimale

Algorithme Minimum(F)
Data: F, un ens de DF sur R
Result: G, une couverture minimale de F
$G := \emptyset$
foreach $X \to Y \in F$ **do**
  $G := G \cup \{X \to X_F^+\}$
foreach $X \to X_F^+ \in G$ **do**
  Soit $G' = G - \{X \to X_F^+\}$
  if $G' \vdash X \to X_F^+$ **then**
    $G := G - \{X \to X_F^+\}$

Algorithme de synthèse (suite)

Sur l'algorithme précédent :
$G' \vdash X \rightarrow X_F^+$ est équivalent à $X_F^+ \subseteq X_{G'}^+$
$\Rightarrow$ cf Propriété 3 slide 153

Le principe algorithmique est le même pour minimiser les parties gauches puis droites des DF

Algorithme de synthèse (suite)

Comment assurer que la décomposition produite est sans perte de jointure ?

Propriété

Soient R un symbole de relation, F un ensemble de DF sur R et $\mathbf{R} = \{R_1, \ldots, R_n\}$ une décomposition de R.
Si $\mathbf{R}$ préserve les DF de F et qu'il existe un schema R_i de $\mathbf{R}$ tel que schema(R_i) est une clé par rapport à F alors $\mathbf{R}$ est une décomposition sans perte de jointure

Algorithme

Algorithme Synthèse(R, F)
Data: F, un ens de DF sur R
Result: Out, une décomposition de R selon F
$Out = \emptyset$;
$F' = Minimum(F)$;
foreach $X \rightarrow Y \in F'$ **do**
    Minimiser X par rapport à F' ;
    Minimiser Y par rapport à F' ;
foreach $X \rightarrow Y \in F'$ **do**
    Soit R_i un symbole de relation avec $schema(R_i) = X \cup Y$;
    $Out+ = R_i$;
if $\nexists R_i$ *tel que* $schema(R_i)$ *soit une clé* **then**
    Soit S tel que $schema(S)$ soit clé par rapport à F ;
    $Out+ = S$;
return(Out) ;

Résultats

Théorême

Synthese(R, F) *retourne une décomposition sans perte de jointure préservant les dépendances de schema(R). Cette décomposition est en 3FN par rapport à F.*

Algorithme de décomposition

Algorithme récursif, représentation par un arbre de décomposition

Algorithme Décompose(R, F)
Data: F, un ens de DF sur R
Result: *Out*, une décomposition de R selon F
if *R est en FNBC par rapport à F* **then**
  return(R) ;
else
  Soit $X \to Y$ une DF non triviale de F tel que $X_F^+ \neq R$;
  Décompose($XY, F[XY]$) ;
  Décompose($R - (Y - X), F[R - (Y - X)]$) ;

Résultats

Plusieurs choix possibles (ligne 4) pour prendre $X \rightarrow Y$
$\Rightarrow$ mène à des décompositions différentes, certaines plus naturelles que d'autres

Théorême

Décompose(*R*, *F****)*** *retourne une décomposition sans perte de jointure de schema(R). Cette décomposition est en BCNF par rapport à F.*

Exemple

Exemple

Soient R un symbole de relation avec schema$(R) = \{A, B, C\}$ et $F = \{AB \to C, C \to A\}$

- *Appliquer l'algorithme de synthèse et l'algorithme de décomposition sur R.*
- *Commenter vos résultats*

Exemple sur la "Gestion de cours"

Soit $\mathcal{U} = \{C, P, H, S, E, N\}$ représentant les attributs Cours, Professeur, Heure, Salle, Etudiant et Note respectivement.
Soient les contraintes suivantes :
$F = \{C \rightarrow P, HS \rightarrow C, HP \rightarrow S, CE \rightarrow N, HE \rightarrow S\}$
Exercice :

- Décomposer $\mathcal{U}$ en FNBC
- Synthétiser $\mathcal{U}$ en 3FN

Outline

Exemples de modèles conceptuels

Le plus connu en France est le *Modèle Entité-Association* (ou Entité-Relation)

- Modèle conceptuel des données (MCD) de la méthode de conception des systèmes d'information MERISE
- se retrouve avec des variantes dans les méthodes de conception de logiciels (e.g. OMT, RUP)

Définit en 1974 !
De nombreuses extensions du modèle EA existent ...
Les aspects `dynamiques, processus` ne sont pas pris par en compte
Du fait du haut niveau d'abstraction, les **langages** n'ont pas été développés
Les **contraintes** restent anecdotiques de part leur puissance structurelle

Intérêts des modèles conceptuels

1. Réduire la "distance sémantique" entre le domaine d'application et sa représentation "informatique"
   - modélisation du monde réél plus naturelle, plus directe
   - dialogue plus aisé entre informaticiens et utilisateurs
   - aucune considération d'implémentation
   - produit souvent des schémas de relation déjà normalisés
2. Représentation graphique des constructeurs
   $\Rightarrow$ très intuitives : facilité d'utilisation, convivialité de la conception
   $\Rightarrow$ Pour autant, syndrome des "flèches/patates" possible

suite

3. Mécanismes d'abstraction pour représenter les données :
   - *Classification* : regrouper les objets similaires dans une "entité" ou un attribut (en gommant les différences entre les objets)
   - *Agrégation* : obtenir une nouvelle entité à partir d'objets/attributs existants

Modèles conceptuels/logiques/physiques

- Un modèle EA est un modèle conceptuel
- Le modèle relationnel est un modèle logique
- Les éléments du langage C++ pour coder les données
  $\Rightarrow$ modèle physique

Les différences des uns par rapport aux autres impliquent une étape de **traduction de schémas**

Modèles conceptuels vs modèles logiques/physiques

Par rapport aux trois composantes du modèle relationnel, les modèles conceptuels de données ont :

- une limite structure/contraintes moins nette
  $\Rightarrow$ Ceci est du en partie au nombre et à l'expressivité des constructeurs dont on dispose dans un modèle conceptuel
- Peu de langages de requêtes

Limites des Modèles conceptuels

1. *Relativisme sémantique* : suivant notre perception d'un domaine d'application, on peut interpréter la *même information* de façon différente
   $\Rightarrow$ pour un problème de modélisation donné, il n'existe pas UNE solution mais beaucoup de solutions possibles, certaines meilleures que d'autres par rapport à des critères objectifs (les DFs) ou subjectifs (le schéma est joli).
2. Représentation graphique des constructeurs $\Rightarrow$ syndrôme des "flèches/patates" : on peut donc faire n'importe quoi ! ! !
3. Plusieurs modèles conceptuels
   - Quelles modèles/extensions sont prises en compte ?
   - Quelles conventions pour les cardinalités ? (typiquement à la UML ou à la Merise)

   $\Rightarrow$ pas un modèle conceptuel le même

EA : Un modèle conceptuel des données

On considère un modèle conceptuel type Entité-Association avec des notations dites "américaines" pour l'interprétation des cardinalités des associations
$\Rightarrow$ Attention : pour ceux qui connaissent la méthode de conception de système d'information MERISE, la sémantique des notations sera différente.
Les constructeurs sont introduits en se référant aux données réelles que l'on modélise

Les types d'entités

Ils sont formés *à partir d'entités ou d'objets du monde réel qui ont des caractéristiques communes, appelées des attributs*.
En clair, on choisit de regrouper dans un seul "concept" des objets dont les similitudes sont plus grandes que les dissimilitudes.

Exemple

on modélise les enfants d'une école. Par exemple on a Emma, Antoine, Claire et Arthur qui ont respectivement 7, 5, 7 et 10 ans. A partir de ces données, on propose un type d'entité `Enfant` *avec quatre attributs :* `nom`, `prénom`, `age` et `école`.

Représentation graphique

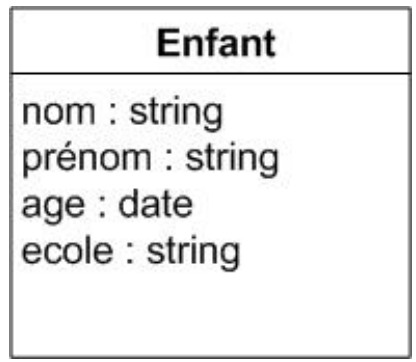

Figure: Un type d'entités

Attributs vs types d'entités

Les ingrédients de base pour concevoir un type d'entités sont les **attributs**
Les attributs peuvent être **monovalués, multivalués ou complexes**.
$\Rightarrow$ parfois une même réalité peut se modéliser avec un attribut ou un type d'entités (cf relativisme sémantique)

Exemple

Si l'on souhaite représenter des informations relatives aux écoles, on peut promouvoir l'attribut `école` au rang de type d'entités avec de nouveaux attributs, `nom`, `adresse`, `directeur`, `capacité`.
De même, l'attribut `prénom` dans `Enfant` pourrait être multivalué.

Représentation graphique

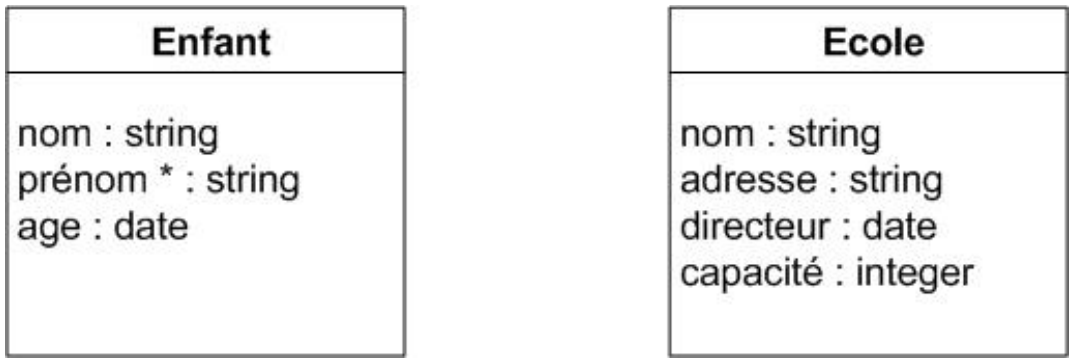

Figure: Deux types d'entités

Les clès dans les types d'entités

récupérées du modèle relationnel !
On permet au concepteur de spécifier un sous ensemble des attributs du type d'entités comme étant un identifiant unique des objets du monde réel représentés par ce type d'entités.
Deux cas :

- il en existe au moins une de façon naturelle
- il n'en existe pas de façon naturelle : on ajoute un attribut (surrogate key) qui fera office de clé.

Exemple

Pour `Enfant`, on ajoute une clé, soit `code` alors que pour `Ecole`, on suppose que `nom` est un identifiant.

On impose un identifiant à chaque type d'entités.

Représentation graphique

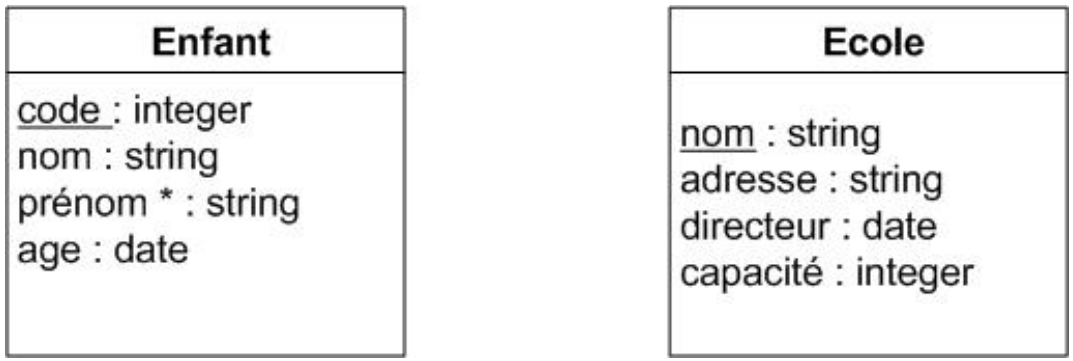

Figure: Deux types d'entités avec clés

Les types d'association

Ils sont identifiés à partir *d'associations existantes entre les objets du monde réel de l'application qui sont devenus des types d'entités*
$\Rightarrow$ nécessite au moins deux types d'entités

Exemple

on souhaite représenter l'association entre les enfants et leur école. Dans ce cas, on crée un nouveau type d'association e.g. `scolariser` *à partir de* `Enfant` *et* `Ecole`*, avec un attribut* `date` *pour représenter la date d'entrée de l'enfant à l'école.*

$\Rightarrow$ Ne nécessite pas de définition de clés, bien que cela soit possible

Représentation graphique

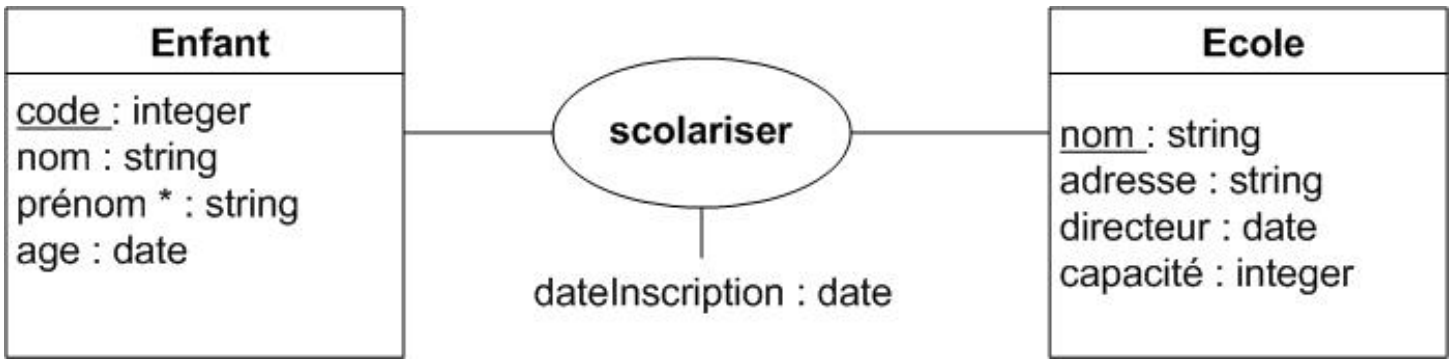

Figure: Un type d'associations

Les types d'association (suite)

Un type d'entité E participant dans un type d'association A est dit avoir un **rôle** dans A.
On parle aussi de **patte** entre E et A.
Les types d'association les plus courant sont **binaires**, ils associent uniquement 2 types d'entités, soit A entre E_1 et E_2.
Au niveau des données mises en jeu par l'association, on a
$A \subseteq E_1 \times E_2$ avec
$E_i = \{\text{objets du monde reel représentés par } E_i\}, i \in \{1, 2\}$

Les types d'association réflexifs

Si $E_1 = E_2$, on parle de **types d'association réflexifs**. Dans ce cas, on est obligé de nommer les pattes de l'association pour comprendre le sens de l'association.

Exemple

Représenter des employés caractérisés par un nom, un salaire et un code et les managers de ces employés, i.e. les employés qui ont à gérer d'autres employés.

1. *en créant deux types d'entités (e.g. Employé et Manager) et un type d'association (e.g. travailler)*
2. *en créant un type d'entité et un type d'association*

Cardinalité des types d'association

Permet d'être plus précis et surtout, de représenter des **contraintes existentielles**, i.e. de participation des entités dans les associations afin de restreindre le produit cartésien représenté par l'association

Exemple

Pour les enfants qui vont à l'école, on peut de façon plus précise dire : un enfant est scolarisé dans une seule école et une école scolarise plusieurs enfants à partir des contraintes de l'application

Pour représenter ce type de contraintes, on va définir des **contraintes de cardinalité maximales**. Ces contraintes valent *un* ou *plusieurs* et se notent sur les pattes des associations

Représentation graphique

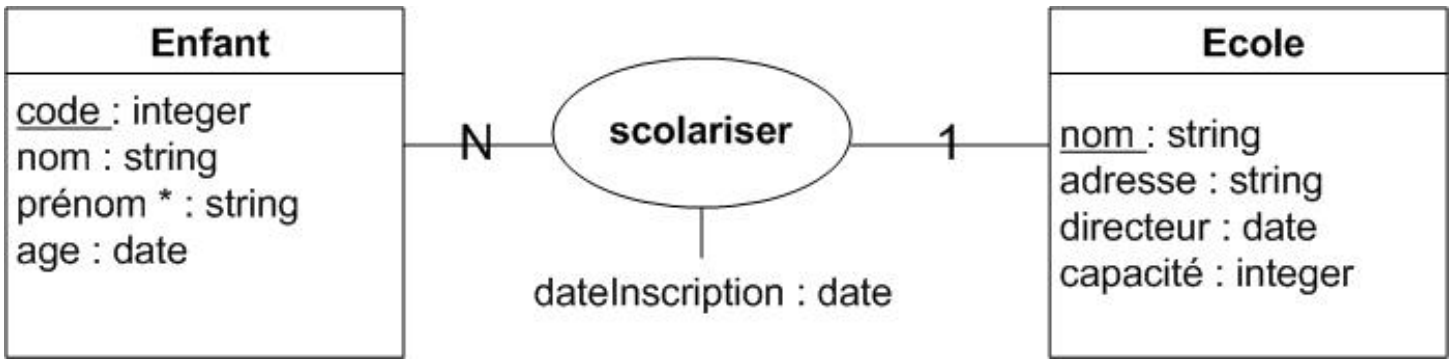

Figure: Contraintes sur un type d'associations

Cardinalité (suite)

Les cardinalités sont des restrictions associées aux rôles.
Les cardinalités maximales peuvent se voir comme des dépendances fonctionnelles entre les attributs clés des types d'entités participants à l'association
On peut aussi définir des **contraintes de cardinalités minimales** : permettent de spécifier des contraintes de participation des entités dans les associations. Valent soit 1 soit 0.
NB : afin de simplifier, ces contraintes ne seront pas utilisées dans ce cours

Les types d'association n-aires

Tout ce qui est dit pour les associations binaires se généralisent facilement pour les associations n-aires
La seule difficulté concerne la sémantique des associations.
Soit A un type d'association entre $E_1, \ldots, E_n$. On considère que chaque E_i représente l'ensemble des objets qu'il modélise.
Soit $A \subseteq E_1 \times E_2 \times \ldots \times E_n$, l'ensemble des instances possibles de A, i.e. un sous ensemble du produit cartésien des entités participantes.

Les types d'association n-aires (fin)

On considère une patte p de l'association, soit entre A et E_i. La cardinalité maximale m pour p est soit 1, soit N.

- si $m = 1$, cela signifie que l'on ne peut pas avoir deux éléments de A tel que ils soient égaux sur tout excepté sur la ième composante, i.e. en E_i
- si $m = N$, il n'y a pas de restriction sur A entre les (n-1) composantes et E_i

$\Rightarrow$ peu utilisés en dépit de leur fort intérêt

Les types d'entités faibles

Ce sont des types d'entités particulier : *ils nécessitent d'autres types d'entités pour assurer leur identification*
$\Rightarrow$ on peut les voir comme une première forme d'association

Exemple

on souhaite représenter par période de 6 mois le poids et la taille des enfants. Par exemple, Emma est née à 3kg850 et mesurait 50 cm, à 6 mois elle faisait 6kg pour 60cm ...
Dans ce cas, on crée un nouveau type d'entité dit faible, , e.g. `DossierEnfant`*, à partir de* `Enfant` *avec comme attributs* `semestre, poids, taille`*.*

Représentation graphique

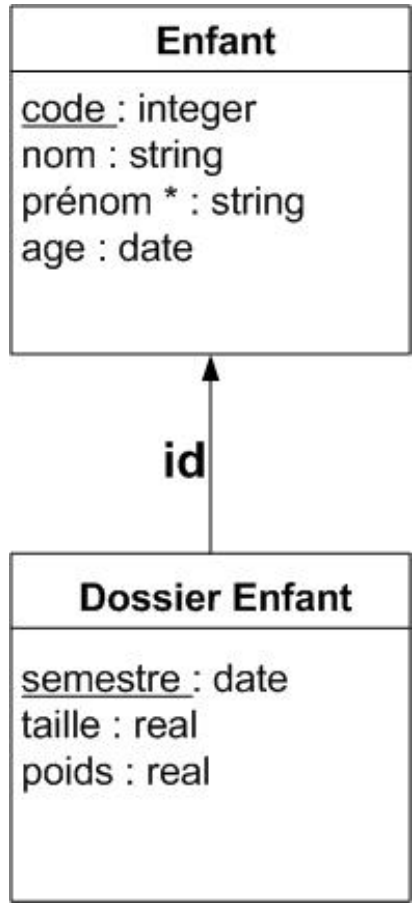

Figure: Un type d'entités faible

Quelques extensions possibles

- Spécialisation/généralisation entre types d'entités ou types d'associations
- Associations généralisées : on permet à une association d'agréger indifféremment des types d'entités et des types d'association
- Types d'entité faible à partir d'un type d'association

Spécialisation/Généralisation

Spécialisation : à partir d'une entité donnée, consiste à représenter une nouvelle entité qui est plus spécifique, i.e. qui dispose d'attributs supplémentaires

Généralisation : à partir de plusieurs entités similaires, consiste à représenter une nouvelle entité qui factorise les propriétés communes aux différentes entités.

Notion d'héritage avec les langages à objet

Représentation graphique

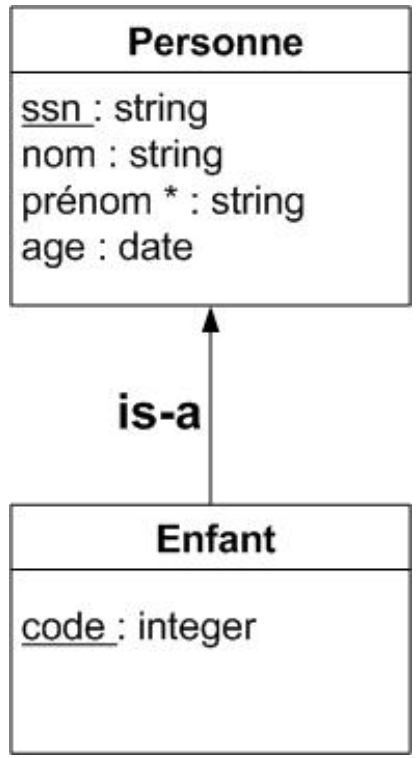

Figure: Lien ISA

Associations généralisées

Permet d'associer indifféremment des types d'entités et des types d'association

On utilise une flèche de type d'association qui associe vers le type d'association qui est associé
$\Rightarrow$ explicite "qui associe qui"

Permet de capturer des contraintes "plus fines"

Représentation graphique

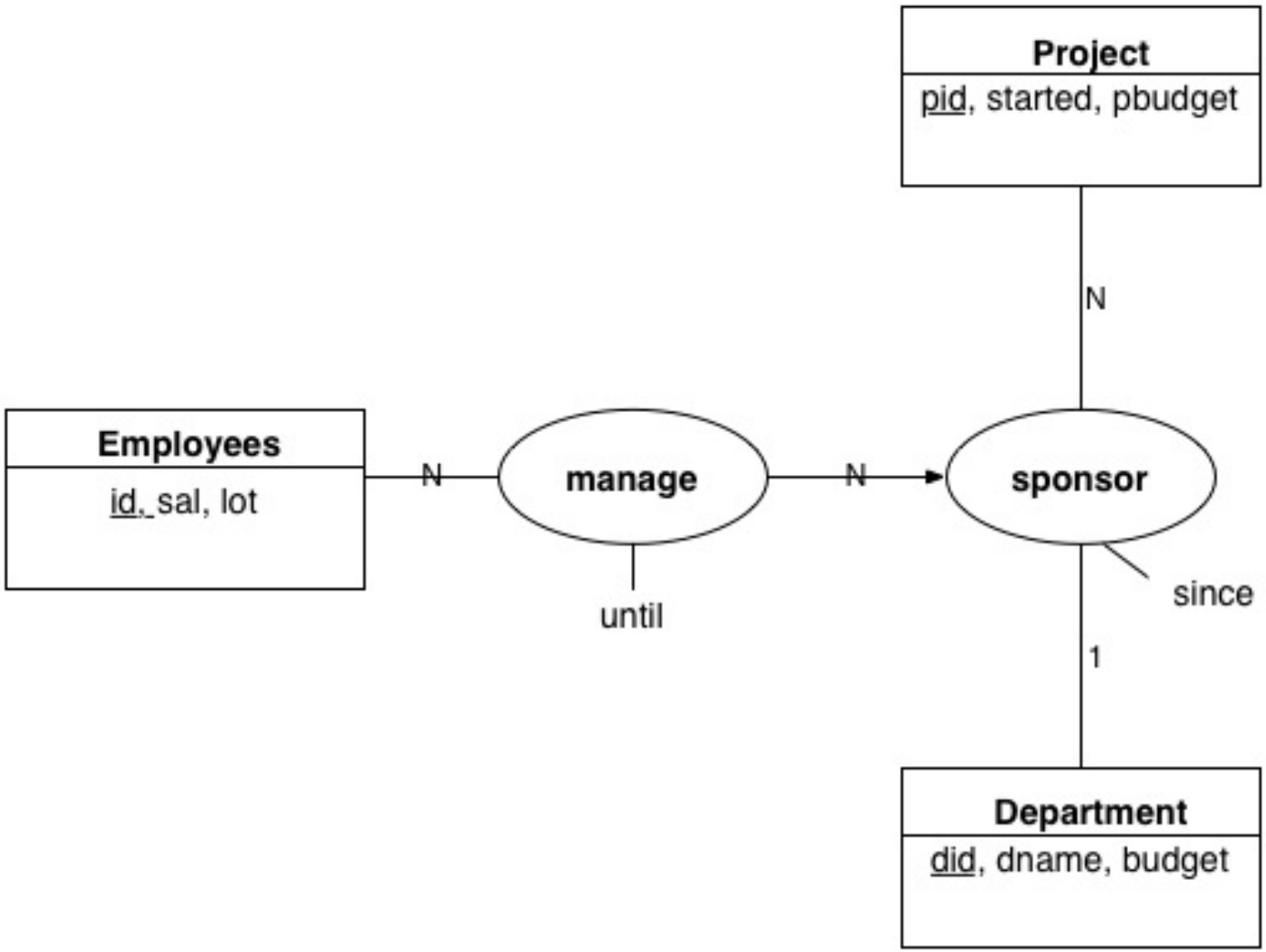

Figure: Association généralisée

Construction d'un schéma conceptuel

Quelles sont les caractéristiques du monde réel à modéliser ?
⇒ Nécessite des entretiens avec les "clients"
⇒ Lecture de documents de spécification quand ils existent
⇒ Facteur d'échelle suivant l'ampleur du projet : besoin de coordonner, de collecter et d'uniformiser les éléments obtenus
Quel est le modèle conceptuel retenu ?
⇒ Si EA, quels constructeurs sont permis ?
⇒ Bien s'entendre sur la signification des constructeurs et des cardinalités

Construction d'un schéma conceptuel (suite)

Conseils **de bon sens** à partir d'un énoncé en langage naturel :

1. Décomposition du texte en propositions élémentaires : **sujet - verbe - complément**
2. Premières structures EA
   - sujet/complément $\rightarrow$ type d'entité ou attribut ;
   - verbe $\rightarrow$ type d'association
3. Intégration des structures EA dans un schéma global
   $\Rightarrow$ se méfier des homonymes, synonymes, informations non pertinentes, non cohérentes

$\Rightarrow$ Le choix des noms est très important
$\Rightarrow$ nécessite des qualités de dialogue de la part des informaticiens avec les personnes du domaine, les autres informaticiens ...

Approche conceptuelle : intérêts

Modélisation du monde réel plus facile de part les nombreux éléments de modélisation disponibles

Production “naturelle” de schémas relationnels normalisés en BCNF !

Les 2 approches coïncident (cf TD)

Traduction entre modèles

Comment obtenir un schéma de bases de données à partir d'un schéma Entité-Association ?
De façon abstraite, on s'intéresse à la traduction de **modèles orientés constructeurs** (e.g. EA) vers des **modèles orientés attributs** (e.g relationnel).
Les contraintes sont :

- implicites dans un modèle conceptuel (orienté constructeurs)
- explicites dans un modèle logique (orienté attribut).

Approche développée

Présentation des différentes traductions possibles et évaluation de chacune d'elles vis à vis de la FNBC
Il va falloir **ajouter des attributs** au niveau relationnel pour représenter correctement certains constructeurs, typiquement les associations.
Les contraintes implicites du modèle conceptuel vont devenir des DF ou des DI au niveau du modèle relationnel
$\Rightarrow$ ensemble de "recettes de cuisine" utile en pratique
Il existe des approches formelles pour faire cela

Traduction évidente

Chaque type d'entités *peut* devenir un schéma de relation, dont le schéma contiendra les attributs mono-valués du type d'entité considéré
Quel est le problème posé par les attributs multivalués ?
Par rapport à ce qui a été vu au niveau du modèle relationnel, qu'est ce qui a changé ?

Exemple

Traduire le type d'entités `Enfant` *en relationnel en considérant que* `prénom` *est soit monovalué soit multivalué.*

Type d'association binaires

Soit A un type d'entités avec a une clé et a_1, a_2 deux attributs. Soit B un type d'entités avec b une clé et b_1, b_2 deux attributs.
On considère un type d'association C entre A et B, ayant deux attributs c_1 et c_2.
On peut considérer que ces trois traductions sont représentatives :

1. Trois symboles de relation A, B, C avec
   $schema(A) = \{a, a_1, a_2\}$, $schema(B) = \{b, b_1, b_2\}$,
   $schema(C) = \{a, b, c_1, c_2\}$
2. Deux symboles de relation A, BC (ou dualement AC, B) avec
   $schema(A) = \{a, a_1, a_2\}$, $schema(BC) = \{b, b_1, b_2, c_1, c_2, a\}$
3. Un symboles de relationt AB, avec
   $schema(AB) = \{a, a_1, a_2, b, b_1, b_2, c_1, c_2\}$

Notez dans certains cas l'apparition de nouveaux attributs et le respect de l'hypothèse de nommage URSA

Type d'association binaires (suite)

Quel type de contraintes peut-on représenter en relationnel ?
Pour les DIs, on obtient :

1. $I = \{C[a] \subseteq A[a], C[b] \subseteq B[b]\}$
2. $I = \{BC[a] \subseteq A[a]\}$
3. $I = \emptyset$

Pour les DFs, on obtient $F = \{a \rightarrow a_1 a_2, b \rightarrow b_1 b_2, ab \rightarrow c_1 c_2\}$

Type d'association binaires : 1-1

Restreindre les associations va naturellement se traduire par l'ajout de nouvelles DF : $F = F \cup \{a \rightarrow b, b \rightarrow a\}$
Quelles sont les clès sur chaque schéma de relation ?
Parmi les 3 traductions possibles, quelles sont celles qui sont en FNBC ?

Type d'association binaires : 1-N

On met le 1 entre A et C.
On obtient : $F = F \cup \{b \rightarrow a\}$
Quelles sont les clès sur chaque schéma de relation ?
Parmi les 3 traductions possibles, quelles sont celles qui sont en FNBC ?

Type d'association binaires : N-N

Pour les trois traductions possibles, l'ensemble F des DFs ne change pas, i.e. on ne peut pas ajouter de nouvelles contraintes.
Quelles sont les clès sur chaque schéma de relation ?
Parmi les 3 traductions possibles, quelles sont celles qui sont en FNBC ?

Autres traductions

Type d'entités faibles
C'est un cas particulier de type d'association binaire N-N.

Exemple

Traduire le type d'entités `DossierEnfant` en relationnel.

Type d'entités spécialisée (issu d'un lien de spécialisation/généralisation)
C'est un cas particulier de type d'association binaire 1-1.

Exemple

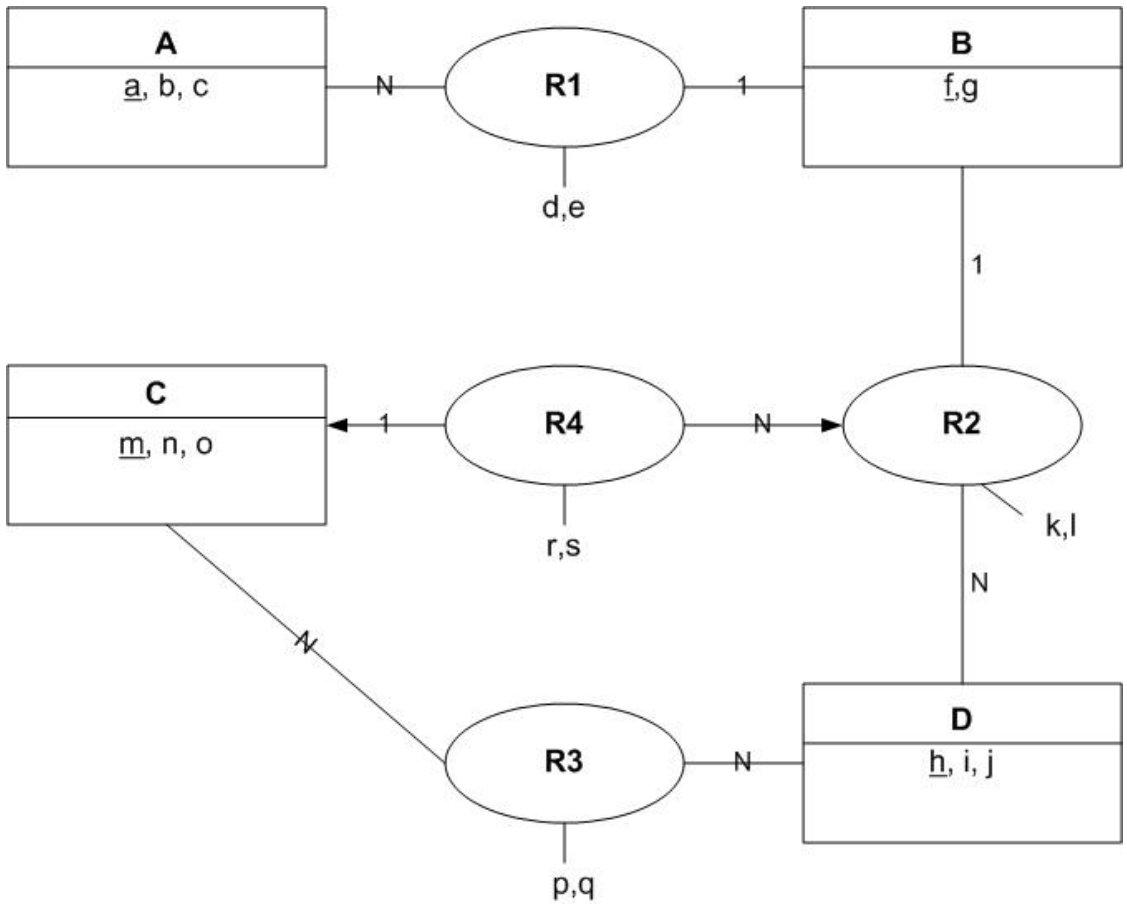

Figure: Exemple

Du relationnel vers EA

$\Rightarrow$ connu sous le terme rétro-ingénierie (ou reverse engineering)
Hypothèses requises pour obtenir un résultat intelligible

- soit toutes les contraintes sont connues, les schémas pouvant être quelconques
- soit aucune contrainte n'est connue, il faut alors des hypothèses de nommage **forte**

Exercice (inverse)

Retrouver le schéma Entité-Association de l'exemple vu en début de cours sur les étudiants, les cours et leur notes (cf diapo 128)

1. Discuter des avantages/inconvénients des différentes alternatives de conception
2. Que pensez vous de l'apparition des valeurs nulles sur chacun des cas ?

Outline

Index des principaux termes