

Fondamentaux de la modélisation des données
3IF – INSA Lyon

Jean-Marc (Calixte) Petit

2016-2017

Sur ce cours

Volume horaire :

9 cours 8 cours, 5 TD , 6 TD 1 examen final

Commentaires/suggestions :

Courriel : jean-marc.petit@insa-lyon.fr

Twitter : @calixtePetit

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**
- ▶ Et surtout, un bel exemple de voir l'**informatique** comme une discipline scientifique et technique

Deux "gros pavés" autour des

1. **Langages**
2. **Contraintes**

3/196

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)
⇒ 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,

4/196

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 Rähkä, "The Design of Relational Databases", Second edition, 1994, Addison-Wesley

5/196

Sommaire

Le modèle relationnel

La structure

Langages de requêtes

L'algèbre relationnelle

Le calcul relationnel

Equivalence des langages relationnels

SQL (Structured Query Language)

Les contraintes

Les dépendances fonctionnelles et les clés

Les dépendances d'inclusion et les clés étrangères

Raisonnement sur les dépendances fonctionnelles

Les relations d'Armstrong

Conception des bases de données

Approche basée sur les contraintes

Approche basée sur le modèle Entité-Association

Le modèle Entité-Association

Conception avec Entité-Association

Traduction entre modèles

Index

6/196

Outline

Le modèle relationnel

- La structure
- Langages de requêtes
 - L'algèbre relationnelle
 - Le calcul relationnel
 - Equivalence des langages relationnels
 - SQL (Structured Query Language)
- Les contraintes
 - Les dépendances fonctionnelles et les clés
 - Les dépendances d'inclusion et les clés étrangères
 - Raisonnement sur les dépendances fonctionnelles
 - Les relations d'Armstrong

Conception des bases de données

- Approche basée sur les contraintes
- Approche basée sur le modèle Entité-Association
 - Le modèle Entité-Association
 - Conception avec Entité-Association
 - Traduction entre modèles

Index

7/196

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-récursif : langage à base de règlesCe 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*.

8/196

Outline

Le modèle relationnel

La structure

Langages de requêtes

L'algèbre relationnelle

Le calcul relationnel

Equivalence des langages relationnels

SQL (Structured Query Language)

Les contraintes

Les dépendances fonctionnelles et les clés

Les dépendances d'inclusion et les clés étrangères

Raisonnement sur les dépendances fonctionnelles

Les relations d'Armstrong

Conception des bases de données

Approche basée sur les contraintes

Approche basée sur le modèle Entité-Association

Le modèle Entité-Association

Conception avec Entité-Association

Traduction entre modèles

Index

9/196

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

<i>Personnes</i>	<i>nss</i>	<i>nom</i>	<i>prenom</i>	<i>age</i>
	12	Aymard	Serge	45
	45	Fenouil	Solange	35

Ici, la première ligne donne les noms aux différentes caractéristiques, appelés *attributs*.

⇒ Version dite **nommée** du modèle relationnel

10/196

Représentation équivalente de la relation Personnes

$\{ \langle 12, \text{Aymard}, \text{Serge}, 60 \rangle, \langle 45, \text{Fenouil}, \text{Solange}, 55 \rangle \}$

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

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

⇒ 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.

11/196

Attributs et domaines

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

Exemple

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

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 $\text{dom}(A)$ le **domaine** de A , i.e. le sous-ensemble de \mathcal{D} dans lequel A peut prendre des valeurs ($\text{dom}(A) \subseteq \mathcal{D}$).

Exemple

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

12/196

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 $\{1, \dots, type(R)\}$ dans \mathcal{U} qui associe des attributs de \mathcal{U} à R **dans un certain ordre**.

On note $schema(R) = \{att_R(1), \dots, att_R(type(R))\}$ 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
 $schema(R) = \{A_1, \dots, A_m\}$, $att_R(i) = A_i$, $i = 1, m$
Un **tuple** sur R est un élément du produit cartésien
 $dom(A_1) \times \dots \times dom(A_m)$
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.

$\langle 12, \text{Aymard}, \text{Serge}, 45 \rangle \in \text{Personnes}$

15/196

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

16/196

Manipulation sur un tuple

Soient R un symbole de relation avec $schema(R) = \{A_1, \dots, A_m\}$, $att_R(i) = A_i (i = 1, m)$, 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 i ème coordonnée de t , i.e. $t(i)$.

→ se généralise à plusieurs attributs

Soit $X = \{att_R(i_1), \dots, att_R(i_n)\} \subseteq schema(R)$. La projection de t sur X , notée $t[X]$, est définie par $t[X] = \langle t(i_1), \dots, t(i_n) \rangle$

Exemple

Soit $t = \langle 12, \text{Aymard}, \text{Serge}, 45 \rangle$. On a :

$t[nom] = t[att(2)] = t(2) = \langle \text{Aymard} \rangle$

$t[nom, prenom] = t[att(2), att(3)] = \langle t(2), t(3) \rangle = \langle \text{Aymard}, \text{Serge} \rangle$

17/196

Symbole de base de données

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

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

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

18/196

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
 $schema(Personne) = \{ nss, nom, prenom, age \}$,
 $schema(Departement) = \{ dep, adresse \}$, $schema(Travaille) = \{ nss, dep, activite \}$.

19/196

Base de données

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

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

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

20/196

Exemple

Soit $\mathbf{d} = \{ Personnes, Departements, Travaillent \}$ 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

21/196

Outline

Le modèle relationnel

- La structure

- Langages de requêtes

 - L'algèbre relationnelle

 - Le calcul relationnel

 - Equivalence des langages relationnels

 - SQL (Structured Query Language)

- Les contraintes

 - Les dépendances fonctionnelles et les clés

 - Les dépendances d'inclusion et les clés étrangères

 - Raisonnement sur les dépendances fonctionnelles

 - Les relations d'Armstrong

Conception des bases de données

- Approche basée sur les contraintes

- Approche basée sur le modèle Entité-Association

 - Le modèle Entité-Association

 - Conception avec Entité-Association

 - Traduction entre modèles

- Index

22/196

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.
⇒ 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.
⇒ dotée d'une *sémantique déclarative*
 - ▶ Datalog
Langage déclaratif à base de règles, capacités d'inférence

⇒ SQL = sucre syntaxique de ces langages

Focus sur les **requêtes conjonctives**

23/196

Proposée par Codd en 1970

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

24/196

Quelques propriétés

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

29/196

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.

30/196

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

31/196

Exemples de jointure

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

					A	D	E
	A	B	C		1	2	3
s_1	2	2	2	s_2	1	1	2
	1	1	1		2	3	1
					3	1	2
	A	B	C		D	E	F
s_3	1	1	1	s_4	1	1	1
	1	2	3		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 $r_1 \bowtie r_2$?

32/196

Le renommage

→ permet par exemple de forcer ou d'éviter des jointures naturelles

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

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

$$\rho_{A \rightarrow B}(r) = \{t \mid \exists u \in r, t[\text{schema}(S) - \{B\}] = u[\text{schema}(R) - \{A\}] \text{ et } t[B] = u[A]\}$$

33/196

La division : l'intuition

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

Exemple

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

	etud	dep		dep
	1	1		1
<i>Inscriptions</i>	1	2	<i>Departement</i>	2
	1	3		3
	2	2		

34/196

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

Propriété

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

35/196

Exemples

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

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

dep
2

etud
2
4

dep
1
2
4

36/196

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 ($\rho_{A \rightarrow B}$) : renomme le nom d'un attribut
 - ▶ Produit cartésien (\times)
 - ▶ Division (\div) : exprime le quantificateur universel (\forall)

37/196

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, ρ
- ▶ l'intersection \cap avec $-$

38/196

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
⇒ à la base de l'**optimisation de requêtes**

39/196

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

⇒ 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.

40/196

Petit exemple introductif

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

		Name	Age	Address	Dept
<i>Etudiants</i>		Vidal	21	Clermont	Computing
		Dupond	35	Paris	Maths
		Durand	45	Lyon	Linguistics
		Dan	34	Lyon	Computing
		Loc	Mgr	Dept	
<i>Departements</i>		Bât A	Ullman	Computing	
		Bât B	Mendelson	Maths	
		Bât C	Dupond	Linguistics	

45/196

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 $\langle x_1 : \textit{Name}, x_2 : \textit{Age} \rangle$,
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 .

46/196

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

47/196

Syntaxe DRC

Symboles autorisés dans une formule

- ▶ Constantes : v_1, \dots, v_n membres de l'ensemble \mathcal{D}
- ▶ Variables : x_1, \dots, 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, \dots, R_n
- ▶ Opérateurs de comparaison : $=$
- ▶ Quantificateurs et connecteurs logiques : $\exists, \forall, \wedge, \vee, \neg$
- ▶ Délimiteurs : $()$ et $,$

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

48/196

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 $v \in \text{DOM}(A)$

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 $\sigma(x) = v$

53/196

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 $\sigma(x) = \sigma(y)$
- ▶ $\langle d, \sigma \rangle \models R(x_1, \dots, x_n)$ ssi $\langle \sigma(x_1), \dots, \sigma(x_n) \rangle \in r$ ($r \in d, r$ 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 $\exists v \in \text{DOM}(A)$ tel que $\langle d, \sigma_{x \mapsto v} \rangle \models G$
- ▶ $\langle d, \sigma \rangle \models \forall x : A(G)$ ssi $\forall v \in \text{DOM}(A), \langle d, \sigma_{x \mapsto v} \rangle \models G$

54/196

Exemples de requêtes

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

57/196

Exemples de requêtes (cont.)

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

58/196

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 $ans(Q, d)$.

59/196

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 :

$$Q_1 : \{ \langle x : A \rangle \mid R(x) \}$$

$$Q_2 : \{ \langle x : A \rangle \mid \neg R(x) \}$$

Pouvez vous écrire une requête équivalente en algèbre relationnelle ?

60/196

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 $ADOM(Q) \cup ADOM(d)$

61/196

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.

$\{ \langle x : nom \rangle \mid \neg Etudiant(x) \}$

$\{ \langle x_1 : nom, x_2 : nom \rangle \mid Etudiant(x_1) \vee Inscrit(x_2) \}$

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

62/196

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

65/196

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 $F_1 - F_4$.

66/196

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

67/196

Requête de calcul relationnel autorisée

- ▶ Une requête autorisée à la forme suivante :

$$\{\langle x_1 : A_1, \dots, x_n : A_n \rangle \mid F(\langle x_1, \dots, x_n \rangle)\}$$

où : F est une formule *bien formée* et *autorisée*,
 A_1, \dots, A_n sont des attributs appartenant à \mathcal{U}
 x_1, \dots, 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.

Théorème

Toute formule autorisée du calcul relationnel est indépendante du domaine.

68/196

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 ?

$$Q_1 = \{ \langle x : nom \rangle \mid \neg Etudiant(x) \}$$

$$Q_2 = \{ \langle x_1 : nom, x_2 : nom \rangle \mid Etudiant(x_1) \vee Inscrit(x_2) \}$$

$$Q_3 = \{ \langle x_1 : nom \rangle \mid \exists x_2 : adresse (Habite(x_1, x_2) \wedge \neg Habite(x_1, Lyon)) \}$$

$$Q'_3 = \{ \langle x_1 : nom \rangle \mid \forall x_2 : adresse (Habite(x_1, x_2) \wedge \neg Habite(x_1, Lyon)) \}$$

Exemple (fin)

$$Q_4 = \{ \langle x_1 : nom \rangle \mid Habite(x_1, Antraigues) \vee Habite(x_1, Lyon) \}$$

$$Q_5 = \{ \langle x : nom \rangle \mid \forall y : nom (\neg Etudiant(y)) \}$$

$$Q'_5 = \{ \langle x : nom \rangle \mid \neg (\exists z : nom (Etudiant(z))) \}$$

$$Q_6 = \{ \langle y : adresse \rangle \mid \forall x : nom (\neg Etudiant(x) \wedge Habite(x, y)) \}$$

Outline

Le modèle relationnel

- La structure

- Langages de requêtes

 - L'algèbre relationnelle

 - Le calcul relationnel

 - Equivalence des langages relationnels

 - SQL (Structured Query Language)

- Les contraintes

 - Les dépendances fonctionnelles et les clés

 - Les dépendances d'inclusion et les clés étrangères

 - Raisonnement sur les dépendances fonctionnelles

 - Les relations d'Armstrong

Conception des bases de données

- Approche basée sur les contraintes

- Approche basée sur le modèle Entité-Association

 - Le modèle Entité-Association

 - Conception avec Entité-Association

 - Traduction entre modèles

Index

71/196

Equivalence des langages

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

⇒ Comment énoncer ce résultat ?

⇒ Esquisse de la preuve

72/196

Notions préliminaires

Deux requêtes Q_1 et Q_2 sur un schéma de BD R sont dites **équivalentes**, noté $Q_1 \equiv Q_2$, si pour toute base de données d sur R , $Q_1(d) = Q_2(d)$.

\Rightarrow quelque soit la base de données, les réponses doivent être les mêmes

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

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

73/196

Théorème

Les trois langages relationnels suivants sont équivalents :

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

Résultat fondamental des bases de données

74/196

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 $H(n+1)$

75/196

SQL : un rapide tour d'horizon

76/196

SQL vs Modèle relationnel

⇒ 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 57)

77/196

La sémantique de SQL

Une relation SQL n'est pas un ensemble de tuples!!!

⇒ 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

$$R \cap (S \cup T) = \{1\} \cap \{1,1\} = \{1\} \text{ et}$$

$$(R \cap S) \cup (R \cap T) = \{1\} \cup \{1\} = \{1,1\}$$

78/196

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

⇒ Chaque SGBD a son propre SQL, typiquement pour tous les aspects hors-normes : built-in fonction, fermeture transitive

⇒ 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)

79/196

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> ]
```

⇒ 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 ⇒ on parle de **vue**.

80/196

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 SQL sont **par défaut** définies sur des multi-ensembles
⇒ pour forcer la sémantique ensembliste, utiliser le mot clé 'DISTINCT' après 'SELECT'

Les opérateurs ensemblistes (UNION, INTERSECT, MINUS) force par défaut la sémantique ensembliste!!!

⇒ pour forcer la sémantique multi ensemble, utiliser le mot clé 'ALL' après 'UNION', ...

⇒ 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	Fenuil	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

83/196

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 préfixe l'attribut par le nom de sa relation, i.e. relation.nom

84/196

Jointure naturelle

Exemple

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

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

\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;
```

87/196

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

88/196

Produit cartésien

Il suffit de ne pas expliciter d'attributs de jointure

```
SELECT *  
FROM  Personnes CROSS JOIN Travaillent
```

ou

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

NB : Si un attribut A apparaît dans les 2 tables, le résultat aura A et A_1 .

89/196

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

90/196

Douceurs syntaxiques : composition de requêtes

Le résultat d'une requête SFW peut être utilisé dans la clause FROM ou WHERE d'une autre requête
⇒ En cas d'ambiguïté, utiliser des parenthèses

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

Exercice

Écrire en SQL l'expression algébrique de la division avec $-$, π , \times (cf. diapo 35)

⇒ **Beaucoup de sucre syntaxique possible avec SQL**, voir cours SGBD en 3IF

91/196

Outline

Le modèle relationnel

- La structure
- Langages de requêtes
 - L'algèbre relationnelle
 - Le calcul relationnel
 - Équivalence des langages relationnels
 - SQL (Structured Query Language)

Les contraintes

- Les dépendances fonctionnelles et les clés
- Les dépendances d'inclusion et les clés étrangères
- Raisonnement sur les dépendances fonctionnelles
- Les relations d'Armstrong

Conception des bases de données

- Approche basée sur les contraintes
- Approche basée sur le modèle Entité-Association
 - Le modèle Entité-Association
 - Conception avec Entité-Association
 - Traduction entre modèles

Index

92/196

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
⇒ on ne peut pas insérer tout et n'importe quoi
⇒ 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.

⇒ *notion de clé*

Un employé ne peut pas être affecté à un numéro de projet qui n'existe pas

⇒ *notion de clé étrangère*

93/196

Dépendances fonctionnelles (DF)

Concept très important, inhérent à la **modélisation des données**
⇒ à la base de la *théorie de la conception des bases de données*.

Intérêts pratiques :

- ▶ permet de définir formellement la notion de **bons schémas**
⇒ 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).

94/196

Exemple (suite)

- ▶ *Certaines informations dépendent de l'existence d'autres informations*

Ex : Le cours 'CS24' de BD dépend de l'existence de Paul.

⇒ **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 connaît un de ses cours et sa note dans ce cours, à moins de permettre les valeurs nulles.

⇒ **Anomalie d'insertion**

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

97/196

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 $X, Y \subseteq \text{schema}(R)$

- ▶ 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 ?

$\text{Donnees} \models id \rightarrow \text{nom, adresse}$

$\text{Donnees} \not\models id \rightarrow \text{note}$

98/196

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).

⇒ Notions théoriques sous-jacentes développées après

99/196

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).

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

Définition

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.**

⇒ évite les problèmes de redondance et d'anomalie identifiés précédemment

100/196

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 ?

101/196

Dépendances d'inclusion (DI)

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

Les DF permettent de spécifier des contraintes intra-relations
⇒ les DI permettent de spécifier des contraintes inters-relations

Les DF généralisent les clés
⇒ les DI généralisent les **clés étrangères**

102/196

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!!!

⇒ l'idée est à nouveau de restreindre les insertions permises en spécifiant 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"

103/196

Syntaxe des DI

ATTENTION : on parle de *séquences d'attributs distincts* pour les DI, à ne pas confondre avec les ensembles.

Stricto sensu, on devrait redéfinir les opérateurs ensemblistes usuels pour les séquences ...

Definition

Soient \mathbf{R} un schéma de base de données R_1 et R_2 deux schémas de relation de \mathbf{R} .

Une dépendance d'inclusion sur \mathbf{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

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

104/196

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

Exemple

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

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

Exercice

Donner une définition des *clés étrangères* en fonction des clés et des DI.

105/196

Règles d'inférence pour les dépendances d'inclusion

Soit I un ensemble de DI sur un schéma de base de données $\mathbf{R} = \{R_1, \dots, R_n\}$

Système d'axiomes de Casanova et al. :

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

Théorème

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

106/196

Le modèle relationnel

- La structure

- Langages de requêtes

 - L'algèbre relationnelle

 - Le calcul relationnel

 - Equivalence des langages relationnels

 - SQL (Structured Query Language)

- Les contraintes

 - Les dépendances fonctionnelles et les clés

 - Les dépendances d'inclusion et les clés étrangères

 - Raisonnement sur les dépendances fonctionnelles

 - Les relations d'Armstrong

Conception des bases de données

- Approche basée sur les contraintes

- Approche basée sur le modèle Entité-Association

 - Le modèle Entité-Association

 - Conception avec Entité-Association

 - Traduction entre modèles

[Index](#)

[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

⇒ **Pré-requis pour aller plus loin**

108/196

Ensemble de DF

Soient F et G deux ensembles de DF sur R .

⇒ 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* : "Étant donné 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

109/196

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 $X \rightarrow Y$ à partir de F en appliquant des règles d'inférence.

- ▶ Règles d'inférence d'Armstrong
 1. Réflexivité : si $Y \subseteq X \subseteq \text{schema}(R)$, alors $F \vdash X \rightarrow Y$
 2. Augmentation : si $F \vdash X \rightarrow Y$ et $W \subseteq \text{schema}(R)$, alors $F \vdash XW \rightarrow YW$
 3. Transitivité : si $F \vdash X \rightarrow Y$ et $F \vdash Y \rightarrow Z$, alors $F \vdash X \rightarrow Z$
- ▶ Notion de preuve :
Soient F un ensemble de DF sur R et $X \rightarrow Y$ une DF sur R .
 F implique $X \rightarrow Y$, notée $F \vdash X \rightarrow Y$, 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 $X \rightarrow Y$.

Equivalence des deux approches

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

Théorème

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

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

$$\mathcal{U} = \{(i, j) \mid i = 1..n, j = 1..m\}$$

$\Rightarrow \mathcal{U}$ représente donc l'ensemble des étudiants

(1, 1)	...	(1, m)
⋮	⋮	⋮
(n, 1)	...	(n, m)

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

114/196

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 :

$$\blacktriangleright F = \{(1, j), (2, j), \dots, (n, j)\} \rightarrow \{(1, j + 1)\} \mid j = 1..m - 1\}$$

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

115/196

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

116/196

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 !

117/196

Fin de la distraction !

118/196

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

La fermeture de X par rapport à F , notée X_F^+ , est définie par :

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

Propriété

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

Algorithme ?

119/196

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^+)

120/196

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.

$CL(F) = \{X \mid X = X_F^+\}$

et $IRR(F)$ ses éléments irréductibles par intersection, i.e.

$IRR(F) = \{X \in CL(F) \mid \text{pour tout } Y, Z \in CL(F)$

$(X = Y \cap Z) \Rightarrow (X = Y \text{ ou } X = Z)\}$

$\Rightarrow schema(R) \in CL(F)$

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

Remarque

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

121/196

Retour sur la notion de clé

⇒ Les clés sont des cas particuliers de *dépendances fonctionnelles*

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

▶ Superclé

X est une *superclé* de R par rapport à F si
 $F \models X \rightarrow \text{schema}(R)$ (ou $X_F^+ = \text{schema}(R)$)

▶ 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

124/196

Projection de F sur un sous ensemble d'attributs

Soient \mathcal{U} l'univers, F un ensemble de DF sur \mathcal{U} et $S \subset \mathcal{U}$

La projection de F sur S , notée $F[S]$, est définie par :

$$F[S] = \{X \rightarrow Y \mid F \models X \rightarrow Y, X \cup Y \subseteq S\}$$

Exemple

Soient $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$ et

$F' = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$ deux ensembles de DFs sur $\{A, B, C\}$.

On considère les sous-ensembles suivants :

$$S_1 = \{A, B\}, S_2 = \{B, C\}$$

Calculer $F[S_1], F[S_2], F'[S_1], F'[S_2]$

125/196

Forme normale

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

Definition

Un attribut $A \in \text{schema}(R)$ 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 $X \rightarrow Y$ 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 $X \rightarrow Y$ de F , X est une clé (pas forcément minimale) de F .

126/196

Le modèle relationnel

La structure

Langages de requêtes

L'algèbre relationnelle

Le calcul relationnel

Equivalence des langages relationnels

SQL (Structured Query Language)

Les contraintes

Les dépendances fonctionnelles et les clés

Les dépendances d'inclusion et les clés étrangères

Raisonnement sur les dépendances fonctionnelles

Les relations d'Armstrong

Conception des bases de données

Approche basée sur les contraintes

Approche basée sur le modèle Entité-Association

Le modèle Entité-Association

Conception avec Entité-Association

Traduction entre modèles

Index

127/196

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

127/196

Relations d'Armstrong pour les DFs

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

Definition

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

128/196

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 :

- ▶ $ag(t_1, t_2) = \{A \in \text{schema}(R) \mid t_1[A] = t_2[A]\}$
- ▶ $ag(r) = \{ag(t_1, t_2) \mid t_1, t_2 \in r\}$

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 $Y, Z \in CL(F)$

$(X = Y \cap Z) \Rightarrow (X = Y \text{ ou } X = Z)$

129/196

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 $B \in (\text{schema}(R) - X)$

130/196

Outline

Le modèle relationnel

La structure

Langages de requêtes

L'algèbre relationnelle

Le calcul relationnel

Equivalence des langages relationnels

SQL (Structured Query Language)

Les contraintes

Les dépendances fonctionnelles et les clés

Les dépendances d'inclusion et les clés étrangères

Raisonnement sur les dépendances fonctionnelles

Les relations d'Armstrong

Conception des bases de données

Approche basée sur les contraintes

Approche basée sur le modèle Entité-Association

Le modèle Entité-Association

Conception avec Entité-Association

Traduction entre modèles

Index

131/196

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.

132/196

Outline

Le modèle relationnel

La structure

Langages de requêtes

L'algèbre relationnelle

Le calcul relationnel

Equivalence des langages relationnels

SQL (Structured Query Language)

Les contraintes

Les dépendances fonctionnelles et les clés

Les dépendances d'inclusion et les clés étrangères

Raisonnement sur les dépendances fonctionnelles

Les relations d'Armstrong

Conception des bases de données

Approche basée sur les contraintes

Approche basée sur le modèle Entité-Association

Le modèle Entité-Association

Conception avec Entité-Association

Traduction entre modèles

Index

133/196

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

⇒ Assurer des **propriétés** pour cette décomposition :

⇒ Algorithmes de normalisation :

- ▶ Algorithme de décomposition
- ▶ Algorithme de synthèse

134/196

Quelles propriétés ?

Quelles sont les propriétés naturelles qu'il faudrait respecter ?

- ▶ Décomposition sans perte de jointure
⇒ aucune donnée n'est perdue
- ▶ Préservation des dépendances
⇒ toutes les dépendances sont conservées

Propriétés pas toujours atteignables

135/196

Définitions

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

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

136/196

Définitions (suite)

Soient R un symbole de relation, F un ensemble de DF sur R et

$\mathbf{R} = \{R_1, \dots, R_n\}$ une décomposition de R .

\mathbf{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 :

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

137/196

Définitions (fin)

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

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

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 ?

138/196

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*

139/196

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

⇒ préserve les dépendances !

140/196

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 $X \rightarrow Y \in F$ **do**

┌ $G := G \cup \{X \rightarrow X_F^+\}$

foreach $X \rightarrow X_F^+ \in G$ **do**

┌ Soit $G' = G - \{X \rightarrow X_F^+\}$

┌ **if** $G' \vdash X \rightarrow X_F^+$ **then**

┌ ┌ $G := G - \{X \rightarrow X_F^+\}$

Algorithme de synthèse (suite)

Sur l'algorithme précédent :

$G' \vdash X \rightarrow X_F^+$ est équivalent à $X_F^+ \subseteq X_{G'}^+$

⇒ cf Propriété 3 slide 120

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

$\mathbf{R} = \{R_1, \dots, R_n\}$ 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

143/196

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**

 | **if** $X \cup Y$ n'apparaît pas déjà dans un schéma de relation **then**
 | Soit R_i un symbole de relation avec $schema(R_i) = X \cup Y$;
 | $Out+ = R_i$;

if $\exists R_i$ 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) ;

144/196

Résultats

Théorème

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

145/196

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 $X \rightarrow Y$ une DF non triviale de F tel que $X_F^+ \neq R$;
 └ Décompose($XY, F[XY]$);
 └ Décompose($R - (Y - X), F[R - (Y - X)]$);

146/196

Résultats

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

Théorème

Décompose(R, F) retourne une décomposition sans perte de jointure de $\text{schema}(R)$. Cette décomposition est en BCNF par rapport à F .

147/196

Exemple

Exemple

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

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

148/196

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

149/196

Le modèle relationnel

La structure

Langages de requêtes

L'algèbre relationnelle

Le calcul relationnel

Equivalence des langages relationnels

SQL (Structured Query Language)

Les contraintes

Les dépendances fonctionnelles et les clés

Les dépendances d'inclusion et les clés étrangères

Raisonnement sur les dépendances fonctionnelles

Les relations d'Armstrong

Conception des bases de données

Approche basée sur les contraintes

Approche basée sur le modèle Entité-Association

Le modèle Entité-Association

Conception avec Entité-Association

Traduction entre modèles

Index

150/196

Outline

Le modèle relationnel

La structure

Langages de requêtes

L'algèbre relationnelle

Le calcul relationnel

Equivalence des langages relationnels

SQL (Structured Query Language)

Les contraintes

Les dépendances fonctionnelles et les clés

Les dépendances d'inclusion et les clés étrangères

Raisonnement sur les dépendances fonctionnelles

Les relations d'Armstrong

Conception des bases de données

Approche basée sur les contraintes

Approche basée sur le modèle Entité-Association

Le modèle Entité-Association

Conception avec Entité-Association

Traduction entre modèles

Index

150/196

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)

Défini en 1974!

De nombreuses extensions du modèle EA existent ...

Les aspects dynamiques, processus ne sont pas pris 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

151/196

Intérêts des modèles conceptuels

1. Réduire la "distance sémantique" entre le domaine d'application et sa représentation "informatique"
 - ▶ modélisation du monde réel 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
 - ⇒ très intuitives : facilité d'utilisation, convivialité de la conception
 - ⇒ Pour autant, syndrome des "flèches/patates" possible

152/196

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

153/196

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
⇒ modèle physique

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

154/196

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
⇒ Ceci est dû en partie au nombre et à l'expressivité des constructeurs dont on dispose dans un modèle conceptuel
- ▶ Peu de langages de requêtes

155/196

Limites des Modèles conceptuels

1. *Relativisme sémantique* : suivant notre perception d'un domaine d'application, on peut interpréter la *même information* de façon différente
⇒ 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 ⇒ syndrome 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)⇒ pas un modèle conceptuel le même

156/196

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

⇒ 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

157/196

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.

158/196

Représentation graphique

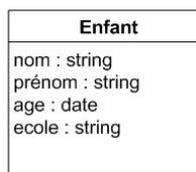


Figure: Un type d'entités

159/196

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**.

⇒ 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.*

160/196

Représentation graphique

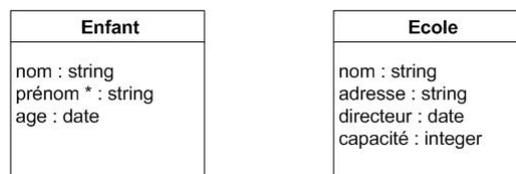


Figure: Deux types d'entités

161/196

Les clés 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.

162/196

Représentation graphique

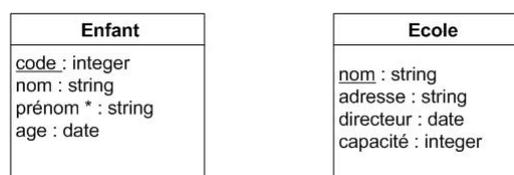


Figure: Deux types d'entités avec clés

163/196

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*
⇒ 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.

⇒ Ne nécessite pas de définition de clés, bien que cela soit possible

164/196

Représentation graphique

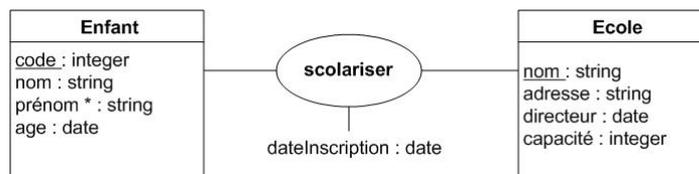


Figure: Un type d'associations

165/196

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

168/196

Représentation graphique

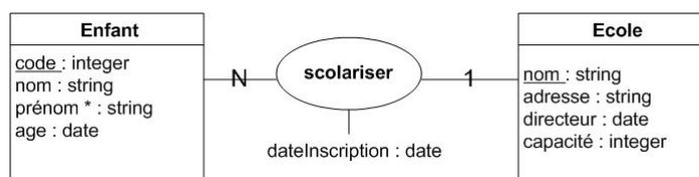


Figure: Contraintes sur un type d'associations

169/196

Représentation graphique

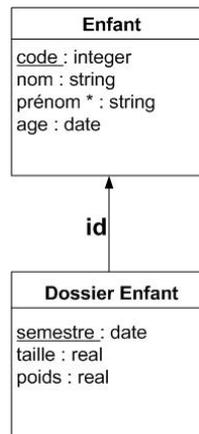


Figure: Un type d'entités faible

174/196

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

175/196

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

176/196

Représentation graphique

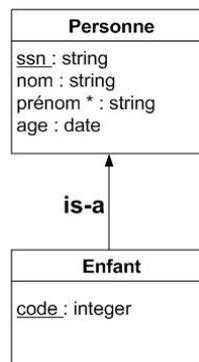


Figure: Lien ISA

177/196

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
⇒ explicite "qui associe qui"

Permet de capturer des contraintes "plus fines"

178/196

Représentation graphique

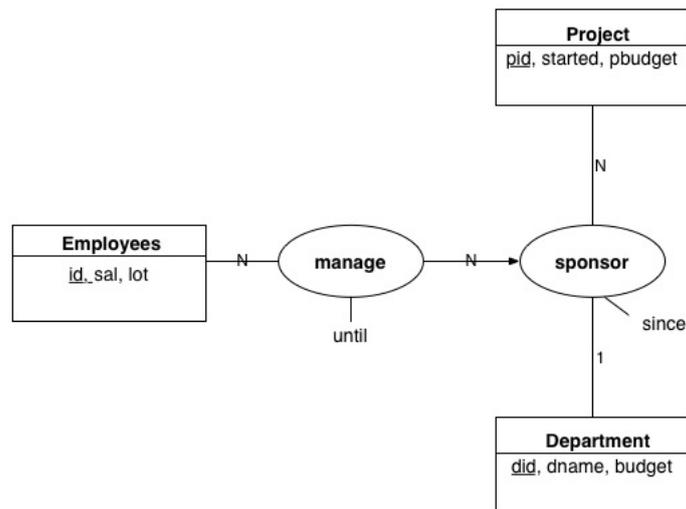


Figure: Association généralisée

179/196

Le modèle relationnel

- La structure

- Langages de requêtes

 - L'algèbre relationnelle

 - Le calcul relationnel

 - Equivalence des langages relationnels

 - SQL (Structured Query Language)

- Les contraintes

 - Les dépendances fonctionnelles et les clés

 - Les dépendances d'inclusion et les clés étrangères

 - Raisonnement sur les dépendances fonctionnelles

 - Les relations d'Armstrong

Conception des bases de données

- Approche basée sur les contraintes

- Approche basée sur le modèle Entité-Association

 - Le modèle Entité-Association

 - Conception avec Entité-Association

 - Traduction entre modèles

[Index](#)

180/196

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

180/196

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 → type d'entité ou attribut ;
 - ▶ verbe → type d'association
3. Intégration des structures EA dans un schéma global
 - ⇒ se méfier des homonymes, synonymes, informations non pertinentes, non cohérentes

⇒ Le choix des noms est très important

⇒ nécessite des qualités de dialogue de la part des informaticiens avec les personnes du domaine, les autres informaticiens ...

181/196

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)

182/196

Le modèle relationnel

- La structure

- Langages de requêtes

 - L'algèbre relationnelle

 - Le calcul relationnel

 - Equivalence des langages relationnels

 - SQL (Structured Query Language)

- Les contraintes

 - Les dépendances fonctionnelles et les clés

 - Les dépendances d'inclusion et les clés étrangères

 - Raisonnement sur les dépendances fonctionnelles

 - Les relations d'Armstrong

Conception des bases de données

- Approche basée sur les contraintes

- Approche basée sur le modèle Entité-Association

 - Le modèle Entité-Association

 - Conception avec Entité-Association

 - Traduction entre modèles

[Index](#)

183/196

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).

183/196

Approche développée

Présentation des différentes traductions possibles et évaluation de chacune d'elles vis à vis de la FNBC

Il faut **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

⇒ ensemble de "recettes de cuisine" utiles 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 relation 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 $F = \{a \rightarrow a_1a_2, b \rightarrow b_1b_2, ab \rightarrow c_1c_2\}$

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 clés 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 clés 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 clés sur chaque schéma de relation ?
Parmi les 3 traductions possibles, quelles sont celles qui sont en FNBC ?

190/196

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.

191/196

Exemple

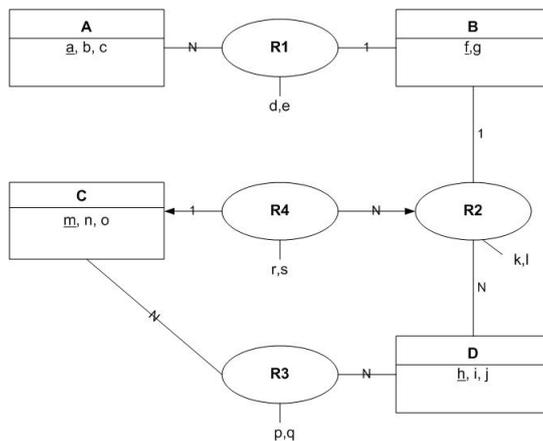


Figure: Exemple

192/196

Du relationnel vers EA

⇒ 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**

193/196

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 95)

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

194/196

Outline

Le modèle relationnel

- La structure

- Langages de requêtes

 - L'algèbre relationnelle

 - Le calcul relationnel

 - Équivalence des langages relationnels

 - SQL (Structured Query Language)

- Les contraintes

 - Les dépendances fonctionnelles et les clés

 - Les dépendances d'inclusion et les clés étrangères

 - Raisonnement sur les dépendances fonctionnelles

 - Les relations d'Armstrong

Conception des bases de données

- Approche basée sur les contraintes

- Approche basée sur le modèle Entité-Association

 - Le modèle Entité-Association

 - Conception avec Entité-Association

 - Traduction entre modèles

Index

195/196

Index des principaux termes

196/196

affectation (de variables à des constantes), 53
algorithme de décomposition (FNBC), 148, 149
algorithme de synthèse (3FN), 142, 146
algèbre relationnelle, 23, 24, 37, 74
arbre de requête, 40
attribut, 10, 12

base de données, 20

calcul relationnel, 23, 45, 47, 74
calcul relationnel autorisé, 60
calcul relationnel de domaine (DRC), 47, 48
calcul relationnel de tuple (TRC), 47
clé, 125
contraintes, 93
couverture (ens. de DFs), 124

datalog, 23, 74
différence, 28
division, 34, 91

196/196

domaine, 12
domaine actif (d'un attribut), 16
domaine actif (d'une base de données), 20
domaine actif (d'une relation), 16
décomposition, 138–141
dépendance d'inclusion (DI), 102, 104–106
dépendance fonctionnelle (DF), 94, 98, 99, 102, 109, 110, 123, 125

ensemble de fermés, 122

Fermeture (Algo. de), 121
fermé (ens. d'attribut), 122, 132
forme normale, 14, 18, 127, 136
forme normale de Boyce-Codd, 100
formule autorisée, 67, 68

interprétation, 53
intersection, 28, 32
Irréductibles, 122

jointure, 30, 32, 87

196/196

langages de requête, 23

modèle relationnel, 8, 77

normalisation, 136

première forme normale (1FN), 14, 18
problème d'implication, 110
produit cartésien, 32, 38, 89
projection, 25, 83
propriétés algébriques, 29, 35, 41

relation, 10, 15
relation d'Armstrong, 129–132
renommage, 33, 84
requête autorisée, 68
requete SQL, 80
requêtes (algébriques), 39
requêtes de calcul (DRC), 52
requêtes récursives, 43

196/196

satisfaction (d'une formule), 54
schema (de relation), 13
selection, 83, 85
SQL, 23, 76–79
structure (relationnelle), 8
symbole (de base de données), 18
symbole (de relation), 13
sélection, 26, 27

théorie de la preuve, 110, 112
théorie des modèles, 110, 111
troisième forme normale, 100
tuple, 15, 17

union, 28
union, intersection, difference, 90

variable, 48
variable libre, 51
variable liée, 51

196/196

variable negative, 65, 66
variable positive, 64, 66

196/196