UFR 919 Ingénierie – module 3I009 cours 9 Conception de bases de données (suite)

- Conception logique
- Dépendances fonctionnelles
- Décomposition de schémas et normalisation
- Formes normales
- Algorithme de décomposition

Normalisation

Normalisation : Méthodologie de conception pour produire un « *bon schéma* » par *décomposition* (descendante) d'un schéma d'origine ou par *génération* (ascendante)

Le schéma produit doit

- *éviter les anomalies de mises-à-jour :* *forme normale*
- *préserver la sémantique du schéma d'origine :* *sans perte d'informations et de dépendances*

Idée :

- On part d'un *schéma de relation R* et d'un *ensemble de dépendances fonctionnelles F* définies sur R (contraintes sémantiques)
- On applique un ensemble de transformations logiques de R en respectant les contraintes définies par F

La redondance vient des DF, il est normal que les DF soient utilisées pour normaliser

Raffinement par décomposition

EMP-PROJ

ENO	ENAME	TITLE	SALARY
E1	J. Doe	Elect. Eng.	40000
E2	M. Smith	Analyst	34000
E2	M. Smith	Analyst	34000
E3	A. Lee	Mech. Eng.	27000
E3	A. Lee	Mech. Eng.	27000
E4	J. Miller	Programmer	24000
E5	B. Casey	Syst. Anal.	34000
E6	L. Chu	Elect. Eng.	40000
E7	R. Davis	Mech. Eng.	27000
E8	J. Jones	Syst. Anal.	34000

⟫ ⟪

PNO	PNAME	BUDGET	DURATION	RESP
P1	Instrumentation	150000	12	Manager
P1	Instrumentation	150000	24	Analyst
P2	Database Develop.	135000	6	Analyst
P3	CAD/CAM	250000	10	Consultant
P4	Maintenance	310000	48	Engineer
P2	Database Develop.	135000	18	Programmer
P2	Database Develop.	135000	24	Manager
P4	Maintenance	310000	48	Manager
P3	CAD/CAM	250000	36	Engineer
P3	CAD/CAM	250000	40	Manager

ENO→ENAME, TITLE, SALARY
PNO → PNAME, BUDGET *ENO, PNO → DURATION, RESP*
Décomposé en EMP(ENO,ENAME, TITLE, SALARY)
PROJECT(PNO, PNAME, BUDGET)
EMP-PROJbis(ENO, PNO, DURATION, RESP)

- Qu'est-ce qu'une bonne décomposition ?
- Est-ce que une décomposition donnée est bonne ?
- Comment d'obtenir une bonne décomposition ?

Problèmes de la normalisation

- **Problème** : Décomposer le schéma (S,F) en plusieurs relations sans "perdre" des informations et/ou des dépendances dans F :
  - Décomposition sans perte d'informations : pour *chaque base de données (BD)* du schéma S qui satisfait F, il doit être possible de la reconstruire (par des jointures) à partir des tables obtenues après la décomposition (par projection).
  - Décomposition avec préservation des dépendances : il doit être possible de vérifier toutes les contraintes définies par F sans faire de jointures (efficacité).
- Impact sur les requêtes ?
  - Le temps d'exécution peut augmenter à cause des jointures nouvelles

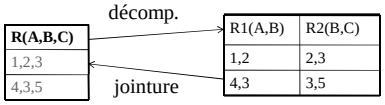
Décomposition de relations

- Un schéma $\{R_1, R_2, \ldots, R_n\}$ est une décomposition du schéma de relation R si $R = R_1 \cup R_2 \cup \ldots \cup R_n$
  et il y a suffisamment d'attributs commun pour joindre tous les Ri. Par exemple, {R1(AB), R2(CD)} n'est pas une décomposition de R(ABCD)
- Pourquoi la décomposition de R(ABC) en R1(AB), R2(BC) est «mauvaise» (s'il n'y a pas de DF) ?
- *Réponse* : Il existe des instances de R où la jointure de R1 et R2 « invente » des n-uplets nouveaux (exemple ci-dessous).
- Est-ce que la décomposition est encore mauvaise quand on sait que R satisfait $B \rightarrow C$?
- *Réponse* : Non (la relation R ci-dessous ne satisfait pas cette DF, voir transparent suivant))

Décomposition SPI

Décomposition Sans Perte d'Information (SPI) :

si une instance *r* de *R* est décomposée en instances r_i de R_1, …, R_n (par projection), on doit pouvoir reconstruire *r* à partir des r_i (par jointure) sans connaissances (nuplets) supplémentaires (ci dessous R satisfait $B \rightarrow C$)



Décomposition sans perte d'information (SPI)

Une décomposition de R en $\{R_1, R_2, \ldots, R_n\}$ est sans perte d'information (SPI) par rapport à un ensemble de DF F ssi:
$\forall r(R)$: si r satisfait F, alors $r = \Pi_{Ri}(r) \bowtie \ldots \bowtie \Pi_{Rn}(r)$

Comment vérifier SPI seulement en regardant les DF:
Algorithme de poursuite (« chase »)

- On a toujours $r \subseteq \Pi_{Ri}(r) \bowtie \ldots \bowtie \Pi_{Rn}(r)$
  (SPI = pas de nuplets en trop)

- Il suffit de prouver l'inclusion dans l'autre sens : on montre que tous les n-uplets t générés par la jointure entre les n-uplets $t_i \in \Pi_{Ri}(r)$ étaient dans r à cause des DF dans F

Tableaux (1/2)

Soit donnée une décompo. de R en $\{R_1, R_2, \ldots, R_n\}$ avec l'ensemble de DF F.

Tableau (représente une instance de R) :

- On veut montrer que $r \supseteq \Pi_{Ri}(r) \bowtie \ldots \bowtie \Pi_{Rn}(r)$
- On prend un n-uplet quelconques dans la jointure et on montre qu'il est dans r
- On définit pour chaque attribut A une constante c_A.
- On crée un tableau (relation) qui contient tous les attributs de R et un nuplet t_i pour chaque schéma R_i tel que
  - $t_i.A = c_A$ si l'attribut A fait partie de R_i t_i est défini sur A
  - $t_i.A$ est une nouvelle valeur (différente de toutes les autres) si l'attribut A ne fait pas partie de R_i t_i n'est pas défini sur A

Tableaux (2/2)

FournisseurProduit(NomF, Adr, NomP, Prix)
est décomposé en
Fournisseur(NomF, Adr) Produit(NomP, NomF, Prix)

Tableau (FournisseurProduit):

	NomF	Adr	NomP	Prix
t1	nom	adr	*b31*	*b41*
t2	nom	*b22*	prod	prix

- t1 est défini sur NomF et Adr
- t2 est défini sir NomF, NomP et Prix

Algorithme de poursuite (1/2)

Soit donnés un tableau T et un ensemble de DF F.

Algorithme de poursuite :

do

a)Choisir une DF $\mathbf{X} \rightarrow Y$ et trouver toutes les lignes T_X de T qui sont *identiques pour tous les attributs dans* **X**.

b) Unification sur Y : *remplacer* dans toutes les lignes T_X et pour tous les attributs A dans Y la valeur $t_i.A$ par

- la constante c_A s'il existe au moins un n-uplet t_j T_X avec $t_j = c_A$
- une valeur (différente de c_A) sinon.

until il existe au moins une ligne complètement défini (succès = décomposition SPI) ou il n'y a plus de remplacement possible (échec).

En pratique, considérer les DF dans un ordre donné, et recommencer à la première lorsqu'on a passé la dernière et pas de ligne complètement définie

Algorithme de poursuite (2/3)

FournisseurProduit(NomF, Adr, NomP, Prix)
est décomposé en
Fournisseur(NomF, Adr) Produit(NomP, NomF, Prix)
F = {NomF → Adr, (NomF, NomP) → Prix}
Tableau de poursuite:

	NomF	Adr	NomP	Prix
t1	nom	adr	*b31*	*b41*
t2	nom	adr	prod	prix

NomF → Adr ⇒ b22 = adr ⇒ t1 ⋈ t2 ∈ r ⇒ décomp. est SPI

Algorithme de poursuite (3/3)

FournisseurProduit(NomF, Adr, NomP, Prix)
est décomposé en
Rel1(NomF, Adr, NomP) Rel2(NomF, Prix)
F = {NomF → Adr, (NomF, NomP) → Prix}
Tableau de poursuite:

	NomF	Adr	NomP	Prix
t1	nom	adr	prod	*b41*
t2	nom	adr	*b32*	prix

NomF → Adr ⇒ b22 = adr ⇒ echec ⇒ décomp. *n'est pas* SPI

Décomposition en 2 et plusieurs relations

Théorème: Soit $\{R_1,R_2\}$ une décomposition de R et F l'ensemble des DF qui s'appliquent à R, alors *la décomposition de R en deux relations $\{R_1,R_2\}$ est SPI ssi l'intersection de R_1 et R_2 est une surclé d'au moins une des deux relations:*

$$\{R_1,R_2\} \text{ SPI} \equiv (R_1 \cap R_2) \rightarrow R_1 - R_2 \textit{ ou } (R_1 \cap R_2) \rightarrow R_2 - R_1$$

Preuve: il suffit de faire le tableau de poursuite

Théorème: Si $\{\mathbf{R_1}, R_2, \ldots, R_n\}$ est SPI de R et $\{S_1, S_2, \ldots, S_m\}$ est SPI de $\mathbf{R_1}$, *alors* $\{S_1, S_2, \ldots, S_m, R_2, \ldots, R_n\}$ est SPI de R.

Décomposition SPD

Décomposition Avec Préservation des DF (SPD) :

- Si (R,F) est décomposé en $R_1, \ldots, R_n$, alors dans chaque instance de R_i on ne peut valider que les DF $X \rightarrow Y$ de F^+ *où R_i contient XY* (F est projeté sur les attributs de R_i : F_i)
- On veut être sûr que les contraintes définies par F sur R peuvent être vérifiées en vérifiant uniquement les DF F_i sur chaque instance R_i

$\Rightarrow$ sinon on serait obligé de *recalculer R (par des jointures)* pour vérifier les DF *perdues !*

Décomposition SPD (1/4)

- { R_1(CodePostal, Rue), R_2(CodePostal, Ville) }
  est une **décomposition *SPI*** de
  R(CodePostal, Ville, Rue) avec
  F={(Ville, Rue) $\rightarrow$CodePostal, CodePostal $\rightarrow$Ville}
  car $(R_1 \cap R_2) \rightarrow R_2 - R_1$ = CodePostal $\rightarrow$ Ville.

- **Mais:** la DF (Ville, Rue) $\rightarrow$ CodePostal ne peut plus être évaluée *efficacement* (sans faire des jointures) : **la décomposition ne préserve pas les dépendances** (SPD).
  => reste à le prouver formellement

Décomposition SPD (2/4)

- **Définition :** Une DF X $\rightarrow$ Y est projetée dans R si R contient XY.
  - Remarque : Une DF non-projeté dans les relations d'une décomposition n'est pas forcément « perdue »: elle peut faire partie de la *fermeture transitive* des DF projetées.

Ex: R(ABCD) décomposée en R1(AB), R2(BC), R3(CD)
et F(A $\rightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ D, D $\rightarrow$ A)

- Il faut considérer F^+ !!!

- **Définition :** Soit F^+_R les DF de F^+ qui se projettent dans R. Une décomposition de R en $\{R_1, R_2, \ldots, R_n\}$ est préserve les dépendances (SPD) dans F si

$$F^+=[F^+_{R1} \cup \ldots \cup F^+_{Rn}]^+$$

Décomposition SPD (3/4)

Théorème : Il suffit de montrer que $F \subseteq [F^+_{R1} \cup \ldots \cup F^+_{Rn}]^+$

Preuve :

- $F^+ \supseteq [F^+_{R1} \cup \ldots \cup F^+_{Rn}]^+$: trivial, car on ne peut pas créer de nouvelles DF lorsqu'on fait une décomposition.
- $F^+ \subseteq [F^+_{R1} \cup \ldots \cup F^+_{Rn}]^+$: il suffit de montrer que toute DF de F peut se déduire des DF projetées : $F \subseteq [F^+_{R1} \cup \ldots \cup F^+_{Rn}]$

Décomposition SPD (4/4)

Entrée: décomposition $D=\{R_1, R_2, \ldots, R_n\}$ et F

Sortie : succès (D est SPD) ou échec

Algorithme : on montre que toutes les DF sont préservées

```
pour toutes les DF X →A ∈ F :
    Z := X
    tant que Z change
        pour toutes les Ri : Z := Z ∪ ([Z∩Ri]+F ∩Ri)
    si A ∉ Z retourne échec
    sinon retourne succès
```

Revient à considérer F+

Exemple : (trivial)

- R(ABCD) et F={A→B, B→C, C→D, D→A}
  R_1(AB), R_2(BC), R_3(CD) est SPD car Z={A,C,B, D} pour A→D

Conception de bases de données

- Conception logique
- Dépendances fonctionnelles
- Décomposition de schémas et normalisation
- Formes normales
- Algorithme de décomposition

1e Forme Normale

- **Définition :** Une relation est en première forme normale (1NF ou 1FN en anglais) quand tous les attributs ont des valeurs atomiques.
- En particulier, une relation 1FN ***ne peut avoir*** une valeur d'attribut qui soit un *ensemble de valeurs* ou un *nuplet.*
- Hypothèse de base des SGBD relationnels.
- Dans les SGBD objet, relationnel-objet et XML, cette contrainte est relâchée.

2e Forme Normale

- L'attribut A *dépend complètement* de X si $X \rightarrow A \in F$ est une DF non-redondante à gauche :
  - $X' \subset X$ implique $X' \rightarrow A \notin F^+$
- A *dépend partiellement* de X sinon

Définition 2FN :

Une relation R est en deuxième forme normale (2FN) par rapport à un ensemble de DF F ssi (si et seulement si) tout attribut de R qui ne fait pas partie d'une **clé** dépend *complètement* de *chaque clé*.

Exemple 2FN

EmpProj(<u>ENO</u>, ENAME, TITLE, <u>PNO</u>, PNAME, RESP)

F = {ENO → (ENAME, TITLE), PNO → PNAME, (ENO, PNO) → RESP}
Une clé : (ENO, PNO)

- **EmpProj n'est pas en 2FN** parce que ENAME, TITLE et PNAME dépendent *partiellement de la clé*
- Ainsi : ENAME et TITLE sont répétés pour chaque produit et PNAME est répété pour chaque employé

Exemple 2FN et redondance

FournisseurProduit(<u>NomF</u>, NomP, Prix, Marque)

F = {NomP → Marque, NomF → (NomP, Prix)}

LA clé : NomF

- **FournisseurProduit est en 2FN :**
  - Aucun attribut non-clé peut en dépendre partiellement d'une clé avec un attribut

Mais : il y a encore des redondances : la marque d'un produit est répétée pour chaque fournisseur.

3ème forme normale

Définition 3FN :

Un schéma de relation R est en troisième forme normale (3FN) par rapport à un ensemble de DF F, *ssi* pour toute DF *non-triviale* $X \rightarrow A$ applicable à R, A est *premier* (fait partie d'une clé) ou X est une surclé *de R.*

FournisseurProduit(<u>NomF</u>, Marque, NomP, Prix)

F = {*NomP → Marque*, NomF→ (NomP, Prix)}
Clé : NomF

FournisseurProduit est en 2FN mais pas en 3FN :
Dans *NomP → Marque*,
Marque n'est pas premier et NomP n'est pas une surclé (Marque dépend *transitivement* de la clé)

On peut montrer que toute relation en 3FN est aussi en 2FN...

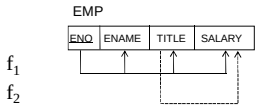
Exemple 3FN + redondance

R(CodePostal, Ville, Rue)
F={(Ville,Rue) → CodePostal, CodePostal →Ville}
Deux clés : (Ville Rue) et (Rue CodePostal)

- **FournisseurProduit est en 3FN :**
  Pas d'attributs non-premier...

Mais il y a encore des redondances : la ville est répétée pour chaque rue.

3FN – Exemple



EMP n'est pas en 3FN à cause de f_2

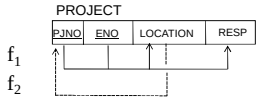
- TITLE → SALARY : TITLE n'est pas une surclé et SALARY n'est pas premier
- le problème est que l'attribut clé ENO détermine transitivement l'attribut SALARY : redondance

Solution : décomposition

- EMP (ENO, ENAME, TITLE)
- PAY (TITLE, SALARY)

Forme normale de Boyce-Codd

Observation : En 3FN, on peut encore avoir des DF transitives où les attributs dépendants sont premiers.



Définition FNBC :
Un schéma de relation R est en forme normale Boyce-Codd (FNBC) par rapport à un ensemble de DF F, si pour toute DF non-triviale $X \rightarrow A$ (de F^+) applicable à R, X est une surclé de R.

Forme Normale de Boyce-Codd

Propriétés de FNBC :

- Tout attribut non-premier dépend complètement de chaque clé.
- Tout attribut premier dépend complètement de chaque clé à laquelle il n'appartient pas.
- Aucun attribut ne dépend d'attributs non-premiers.

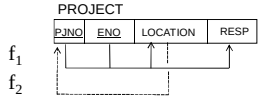
Remarques :

- Toute relation à 2 attributs est FNBC

FNBC = plus aucune redondance dûe aux DF

FNBC – Exemple

- Supposons la relation PROJECT où chaque employé sur un projet a un lieu et une responsabilité unique pour ce projet et il y a un seul projet par lieu



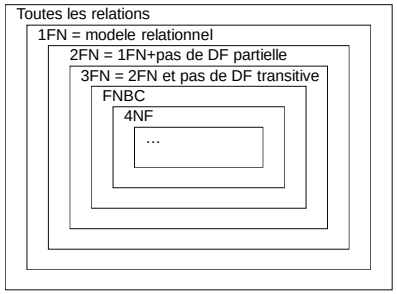
PROJECT est en 3FN mais pas en FNBC

Décomposition :

PR (PJNO, ENO, RESP), PLOC (LOCATION, PJNO)

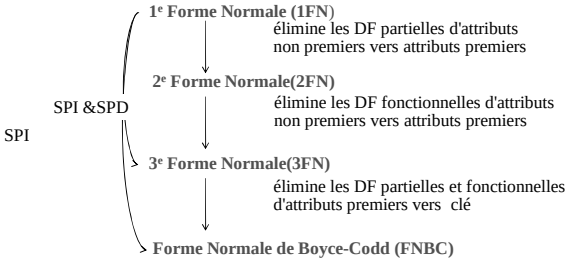
Cette décomposition évite toutes les anomalies mais n'est pas SPD ! (on a perdu la DF (PJNO ENO) → LOCATION)

Formes normales



Formes normales et DF

1FN élimine les attributs non atomiques



(la 4è forme normale implique d'autres contraintes que les DF…)

Conception de bases de données

- Conception logique
- Dépendances fonctionnelles
- Décomposition de schémas et normalisation
- Formes normales
- Algorithme de décomposition

Algorithme de passage en 3FN

Entrée : R(A_1,A_2,…,A_n) et un ensemble minimal de DF F

Sortie : une décomposition SPI et SPD {R_1,R_2,…,R_n} où tous les R_i sont en 3FN

Algorithme :

1.Regrouper les DF qui ont même partie gauche

2.Créer un schéma de relation R_i avec tous les attributs de chaque groupe de DF

3.Si aucune clé de R n'apparaît dans un R_i existant, rajouter un schéma de relation formé par les attributs d'une clé de R

4.Éliminer les schémas de relation inclus dans d'autres

Décomposition en 3FN

Exemple : FournisseurProduit(NomF, Marque, NomP, Prix)
avec F = **{ NomP→ Marque**, NomF→ (NomP, Prix) }

1.Regroupement : rien à faire

2.Création de

- Fournisseur(NomF, NomP, Prix) avec F1= NomF→ (NomP, Prix)
- Produit(NomP, Marque) avec F2={NomP → Marque}

3.Création d'une table pour la clé NomF : pas nécessaire

4.Élimination de schémas inclus dans d'autres schémas : rien à faire

Algorithme de passage en FNBC

Entrée : R(A_1,A_2,…,A_n) et F un *ensemble minimal* de DF F

Sortie : une décomposition SPI {R_1,R_2,…,R_n} où tous les R_i sont en FNBC (décomposition pas forcément SPD)

Algorithme :

1. S:={R}

2. **tant** qu'il existe un $R_i(Y)$ dans S et une DF non-triviale

$X \rightarrow A$ dans $[R_i]^+_F$ telle que X n'est pas surclé de R_i:

$S := (S - R_i) \cup R_j(AX) \cup R_k(Y \backslash A)$

(on décompose Ri en {Rj, Rk})

(noter que c'est bien SPI)

Exemple de décomposition FNBC

Soient

- EMP(ENO, ENAME, TITLE, PNO, PNAME, RESP)
- *F* = {ENO → ENAME , ENO → TITLE => on peut regrouper),
  PNO → PNAME,
  (ENO PNO) → RESP}

- EMP n'est pas en FNBC, car ENO et PNO ne sont pas surclés, donc ENO → (ENAME TITLE) et PNO → PNAME posent problème

Exemple de décomposition FNBC

Commençons avec

D0 = {**EMP(ENO, ENAME, TITLE, PNO, PNAME, RESP)**}

Itération 1

- prendre une des DF qui posent problème : ENO → ENAME TITLE

D1 = {R_1, R_2} où

- **R_1(ENO, PNO, PNAME, RESP)** avec F1 = { PNO → PNAME, (ENO PNO) → RESP}
- **R_2(ENO, ENAME, TITLE)** avec F2 = {ENO → (ENAME TITLE) }

R_2 est en FNBC, mais pas R_1

Exemple de décomposition FNBC

Itération 2

- *D1* contient R_1(ENO, PNO, PNAME, RESP) qui n'est pas en FNBC
- prendre une des DF qui posent problème : PNO → PNAME

D2 = {R_2, R_3, R_4} où

- R_3(ENO, PNO, RESP) avec F1 = { (ENO PNO) → RESP }
- R_4(PNO, PNAME) avec F1 = { PNO → PNAME }

R_2, R_3, R_4 sont alors en FNBC (et on a pu préserver les DF, ce qui n'est pas toujours le cas)