



# ESIR1 BD

## Bases de données

### création

Olivier Ridoux

# Plan

- Création de tables
  - clés
  - index
- Création de vues

# Création de tables

# Clés (1)

- **Clé** : attributs d'une table qui déterminent tous les autres attributs
- Déclaration de clés

- **CREATE TABLE Nom-tab**

- (...

- Nom-attr<sub>i</sub> Dom-attr<sub>i</sub> PRIMARY KEY,**

- ...);**

**Nom-attr<sub>i</sub> → autres-attributs**

**Nom-attr<sub>i</sub> ≠ NULL**

## Clés (2)

- Déclaration de clés
  - **CREATE TABLE fournisseur**  
( numf INT PRIMARY KEY,  
nomf TEXT );
  - **CREATE TABLE produit**  
( nump INT PRIMARY KEY,  
nomp TEXT );

numf	nomf
12	"Corelli"
37	"Pirastro"

nump	nomp
132	"Crystal"
74	"Alliance"
112	"Synoxa"
37	"Eudoxa"
13	"Chorda"

# Clés (3)

- Clés étrangères

– **CREATE TABLE Nom-tab**

**(...**

**Nom-attr<sub>i</sub> Dom-attr<sub>i</sub>**

**REFERENCES Autre-tab,**

**...);**

# Clés (4)

- Clés étrangères
  - **CREATE TABLE fournisseurde**  
**(nump INT PRIMARY KEY,**  
**numf INT REFERENCES fournisseur**  
**);**

nump	numf
132	12
74	12
112	37
37	37
13	37

# Remarque – clés (1)

- **Clé** : attributs (colonnes) d'une table qui déterminent tous les autres
- Notion **a priori**
  - commandée par la logique des données
  - décidée indépendamment d'une donnée particulière
  - ne dépend pas fortuitement des données



## Remarque - clés (2)

- **Clé primaire**

- la table où la clé est primaire **déclare** des valeurs de clé

- **Clé étrangère**

- la table où une clé est étrangère **utilise** des valeurs de la clé qui doivent être déclarées dans la table où la clé est primaire

# Contraintes d'intégrité

- Vérifiées à chaque mise-à-jour
  - PRIMARY KEY
    - Nom-d-attribut → autres-attributs
    - Nom-d-attribut  $\neq$  NULL
  - REFERENCES
    - Vérifie que clé primaire existe
  - CHECK
  - NOT NULL
  - UNIQUE

# Index

- Améliorer les performances
- Remplacer accès séquentiel par accès direct

# Calcul des jointures (1)

- $R_1 \bowtie R_2$
- Pour chaque ligne  $r_1$  de  $R_1$  faire  
pour chaque ligne  $r_2$  de  $R_2$  faire  
si  $r_1.attr = r_2.attr$  alors ajouter( $r_1, r_2$ )
- **Coût =  $taille(R_1) \times taille(R_2) \approx taille(R)^2$**
- **Trop naïf  $\rightarrow$  trop inefficace**

## Calcul des jointures (2)

- **Trier**  $R_1$  et  $R_2$  sur l'attribut commun attr

$r_1$  = 1ère ligne de  $R_1$

$r_2$  = 1ère ligne de  $R_2$

jqà fin de  $R_1$  ou de  $R_2$  faire

si  **$r_1.attr < r_2.attr$**

alors  $r_1$  = ligne suivante de  $R_1$

sinon si  **$r_1.attr > r_2.attr$**

alors  $r_2$  = ligne suivante de  $R_2$

sinon ajouter( $r_1, r_2$ )

# Calcul des jointures (3)

- **Coût = coût tri( $R_1$ ) + coût tri( $R_2$ )**  
**+ taille( $R_1$ ) + taille( $R_2$ )**  
 $= \text{taille}(R_1) \times \log \text{taille}(R_1)$   
 $+ \text{taille}(R_2) \times \log \text{taille}(R_2)$   
 $+ \text{taille}(R_1) + \text{taille}(R_2)$   
 $\approx \text{taille}(R_1) \times \log \text{taille}(R_1)$   
 $+ \text{taille}(R_2) \times \log \text{taille}(R_2)$   
 **$\approx \text{taille}(R) \times \log \text{taille}(R) \ll \text{taille}(R)^2$**

## Calcul des jointures (4)

- Supposons l'existence d'une fonction

**$\text{index}_{\text{attr} \times R_2} : \text{attr} \rightarrow \text{ligne de } R_2$**

- Pour chaque ligne  $r_1$  de  $R_1$

$r_2 = \text{index}_{\text{attr} \times R_2}(r_1.\text{attr})$

ajouter( $r_1, r_2$ )

- **$\text{Coût} = \text{taille}(R_1) \times \text{coût index}_{\text{attr} \times R_2}(\text{attr})$**

**Créer des index efficaces !**

# Index (1)

- **Index**<sub>attr×R</sub>

$a \rightarrow$  position de  $(a, \dots)$  dans  $R$

- Fonction de hachage :  $h$

- Insert  $(a, \dots)$  dans  $R$

$pos = h(a)$

insère  $(a, \dots)$  en position  $pos$  dans  $R$

- Lookup  $a$  dans  $R$

$pos = h(a)$

retourne contenu de la position  $pos$  de  $R$



## Index (2)

- Collision
  - $a \neq b$ , et  $h(a) = h(b)$
  - $(a, \dots)$  et  $(b, \dots)$  en même position dans R
- Efficacité
  - calcul de  $h$
  - nombre de collisions

# Transformations de requête (1)

- Remarques

$$\sigma_{Q_1}(R_1 \bowtie R_2) = \sigma_{Q_1}(R_1) \bowtie R_2$$

$$\text{taille}(\sigma_{Q_1}(R)) \ll \text{taille}(R)$$

- Donc

$$\begin{aligned} \text{coût } \sigma_{Q_1}(R_1) \bowtie R_2 \\ \ll \end{aligned}$$

$$\text{coût } \sigma_{Q_1}(R_1 \bowtie R_2)$$

**Formalisé grâce à AR !**

# Transformations de requête (2)

- $R_1 \times (R_2 \times R_3) = (R_1 \times R_2) \times R_3$
- $\Pi_P(\Pi_{PQ}(R)) = \Pi_P(R)$
- $\sigma_P(\sigma_Q(R)) = \sigma_{P \wedge Q}(R)$
- $\sigma_Q(\Pi_P(R)) = \Pi_P(\sigma_Q(R))$   
si  $Q$  ne porte que sur  $P$
- $\sigma_Q(R_1 \text{ op } R_2) = \sigma_Q(R_1) \text{ op } \sigma_Q(R_2)$   
si  $\text{op}$  est  $\cup$  ou  $\setminus$

...

création

# Création de vues

# Notion de vue (1)

- **Présentation d'un schéma spécialisé pour une application ou un groupe d'utilisateurs**
- CREATE VIEW Nom-vue  
AS SELECT ...

# Notion de vue (2)

- A priori intensionnelle...

...mais éventuellement, vue  
extensionnelle (**matérialisée**)

– cache de calculs intermédiaires

# Pourquoi faire ?

- Indépendance / schéma
- Protection des données
- Consolidation des données
- Structuration des requêtes

## Remarque – notion de vue

- Vue = perte d'information
- Il est possible que
  - $BD_1 \neq BD_2$  et  $\text{vue}(BD_1) = \text{vue}(BD_2)$
- Impossible de revenir de vue à BD





# Indépendance / schéma consolidation des données

- Schéma intermédiaire entre schéma général et applications
- Masque les détails du schéma général
- Reconstitue des données qui sont éparpillées dans le schéma général
  - jointures répétitives



# Protection des données

- Ne montre que les données pertinentes pour une famille d'applications
- **Bienfait** de la perte d'information
  - assure que les données masquées ne peuvent être reconstituées

# Structuration des requêtes

- Donner des noms à des requêtes
- Factoriser des expressions

# Mise à jour de vue

- **Inconvénient** de la perte d'information
- Cas favorable
  - seulement 1 table, avec restriction
- Sinon, vue en **lecture seule**

***View Update Problem***

# Mise à jour des sources d'une vue

- Une vue dépend de relations sources, le FROM...

...que faire si une source est modifiée ?

- Si vue intensionnelle, alors 0 problème...

...sinon rafraichir la vue

- à chaque modification ?
- non plutôt à chaque lecture de la vue

# En image – mises à jour

maj

sources

**View Update Problem !**

**CREATE VIEW**  
**SELECT ...**  
**FROM sources**  
...

vue

maj

création

# Conclusion

- Recherche d'efficacité pour passage à l'échelle
  - clés et index
- Recherche de protection pour différents rôles
  - vues
- Recherche de sécurité pour mise à jour
  - contraintes d'intégrité