

Bases de données avancées Indexation

Sarah Cohen-Boulakia
Laboratoire de Recherche en Informatique
Université Paris Sud
<http://www.lri.fr/~cohen/BD/BD.html>

1

Un exemple pour ce chapitre

- Schéma : EMP(Nom, Age, Salaire)
- Instance : éléments quantitatifs sur le stockage
 - 1 000 000 (un million) d'employées
 - Un enregistrement : 120 octets
 - Un bloc = 4K (34 enregistrements)
 - Fichier de données (table) = 30 000 blocs = 120 Mo
- Modèle de coût
 - B = nb pages de données
 - R = nb enregistrements par page
 - D : tps moyen de transfert

2

Les opérations de recherche

- Recherche par clé
 - attention : pas nécessairement la clé primaire de la relation
 - Nom='Dupont'
- Recherche par préfixe
 - Nom like 'Dupont%'
- Recherche par intervalles
 - Age > 18 and Age < 65
- Autre opérations
 - insertion, suppression (sur critères)

3

On gagne énormément avec un index

Coût des opérations

- Rappel
 - B : nb pages; D : temps transfert
- Coût pour un fichier

non trié	versus trié sur l'attribut
• Parcours : BD	versus $D \log_2 B$ (dichotomie)
• Recherche	
▪ (=) : BD	versus $D \log_2 B$
▪ ([]) : BD	versus $D \log_2 B$
• Insertion (en fin) : 2D	(vs recherche + décalage)
• Suppression : (B+1) D	(vs recherche + compactage)

4

Les index : types et propriétés

Index

- Clé de recherche (1 att ou une liste triée d'att)
 - AGE
 - Attention : clé de recherche \neq clé d'une relation
 - Index = fichier
 - Enregistrement d'un index = entrée de l'index
 - Recherche = accès index puis accès direct aux données
La recherche d'une clé dans l'index doit être rapide et elle sera suivie d'un accès direct à l'enregistrement
 - Un index accélère l'accès au fichier de données
- Attention : coût de l'index
- Temps de création et maintenance
coût en temps et en espace

Types d'index

- Entrée d'index (notée k^*)
- Contient toujours une valeur k de la clé de recherche
- 3 types d'index
 - Type 1 : $k^* = \text{enregistrement}$ (trier en fonction d'une clé)
 - Type 2 : $k^* = (k, \text{rid})$ // record id (Age, adresse)
 - Type 3 : $k^* = (k, \text{liste de rid})$ (Age, liste de toutes les adresses)
- Organisation des entrées
 - Triées sur la clé de recherche
 - Dans une table de hachage, dans un arbre de recherche...

Structure d'index trié toujours →

Exemple : Type 2

Adresses physiques sur le DDur

	Nom	Age	Salaire
@1	Sylvie	23	45000
@2	David	45	78000
@3	Paul	57	67090
@4	Julie	19	24000
@5	Seb	21	67000
@6	Paul	23	76900
.....			

Disque Dur

k^*	
19	@4
21	@5
23	@1
23	@6
45	@2
57	@3

Fichier d'index
Clé d'index k : Age

Exemple : Type 3

Adresses physiques sur le DDur

	Nom	Age	Salaire
@1	Sylvie	23	45000
@2	David	45	78000
@3	Paul	57	67090
@4	Julie	19	24000
@5	Seb	21	67000
@6	Paul	23	76900
.....			

Disque Dur

k*

19	@4
21	@5
23	@1, @6
45	@2
57	@3

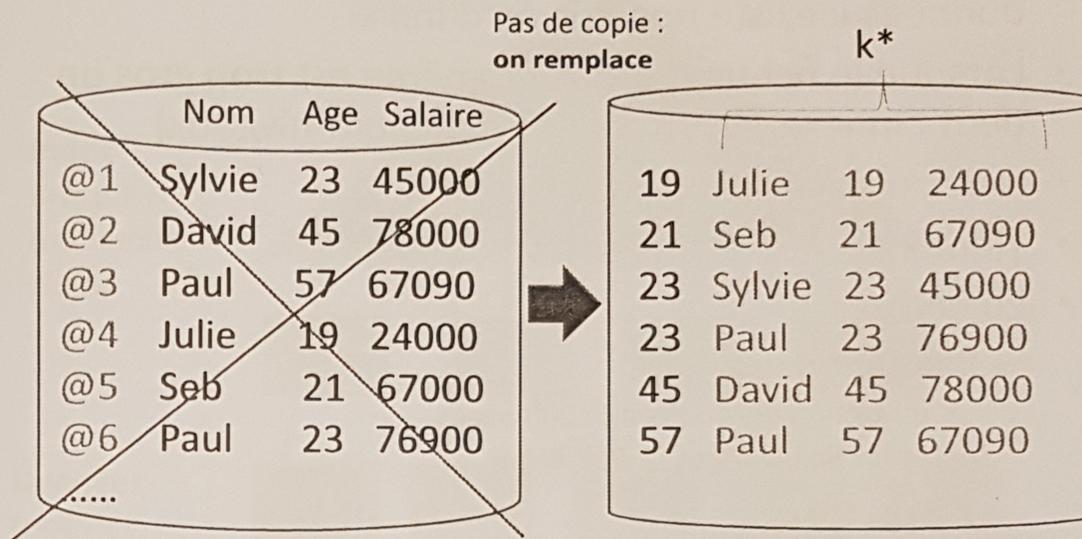
Fichier d'index
Clé d'index k : Age

9

trier sur le disque directement

Exemple : Type 1

L'index contient toute l'info (triée sur la clé d'index)



Disque Dur

Clé d'index k : Age

10

Index groupant

- Lorsque l'index et le fichier de données sont organisés de façon identique relativement à une clé
- Un seul index groupant par fichier
- Impact sur la recherche par clé et par intervalle
- Quel type d'index est toujours groupant ?

Attention : index groupant est très couteux à maintenir !

11

Exemple : Index groupant sur Age (type 2 ici)

	Nom	Age	Salaire
@1	Julie	19	24000
@2	Seb	21	67000
@3	Sylvie	23	45000
@4	Paul	23	76900
@5	David	45	78000
@6	Paul	57	67090
.....			

Disque Dur

Groupant : Les données ont été triées sur le disque

k*

19	@1
21	@2
23	@3
23	@4
45	@5
57	@6

} rid

Fichier d'index
Clé d'index k : Age

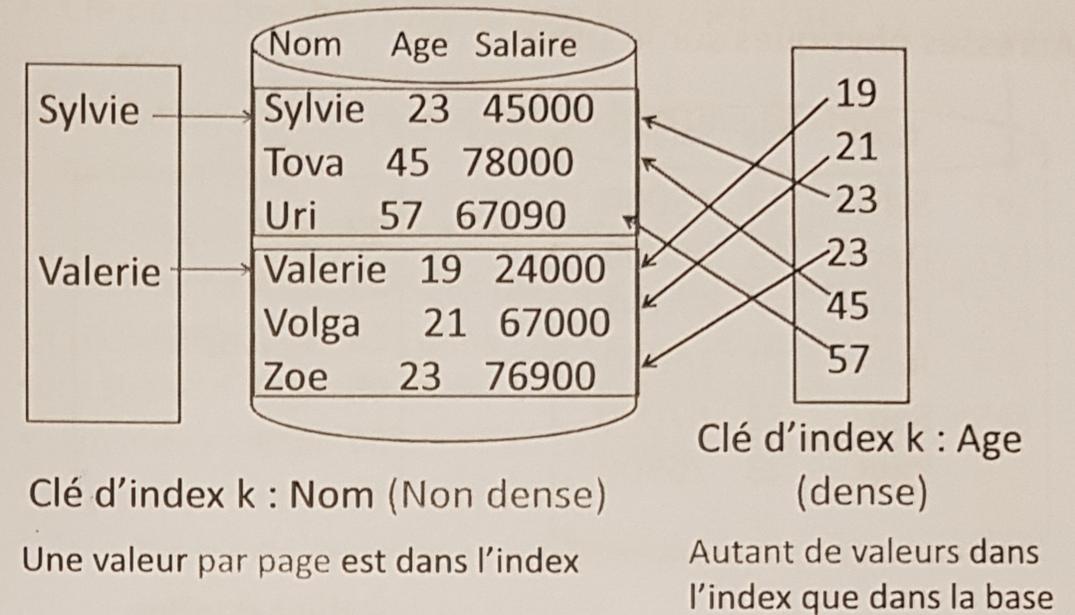
12

Index dense et non dense

- Index dense : pour chaque n-uplet du fichier de données il existe une entrée d'index
- Lorsque le fichier d'index lui-même est trop gros on peut l'indexer à son tour (index multi-niveaux)
- Index non dense → index groupant
- $|\text{index dense}| \geq |\text{index non dense}|$
- En chiffres
 - Dans notre exemple
 - Index non dense (Nom) : 200 pages
 - Index dense (Age) : 2 942 pages

13

Exple : dense (Age), non dense (Nom)



14

Structures pour un index

Non groupant : table pas triée

15

Structures pour un index

- Un fichier trié
- Index bitmap
 - Considère toutes les valeurs possibles de l'attribut indexé
 - Table de n (nb de valeurs possibles) * l (nb de lignes) bits
 - Très utile si les valeurs de l'attribut sont « catégorisées »
- Une table de hachage
 - Avantage : recherche (=)
 - Typiquement utilisée pour les index uniques (clé primaire)
- Un arbre B
 - Avantage : range (inégalités)
 - Arbre parfaitement équilibré (*balanced*)
 - Ordre d : nombre max d'éléments d'un nœud
 - Ordre de l'arbre lié à la taille de la page
 - Nombre de niveaux (empirique) : 3 à 4

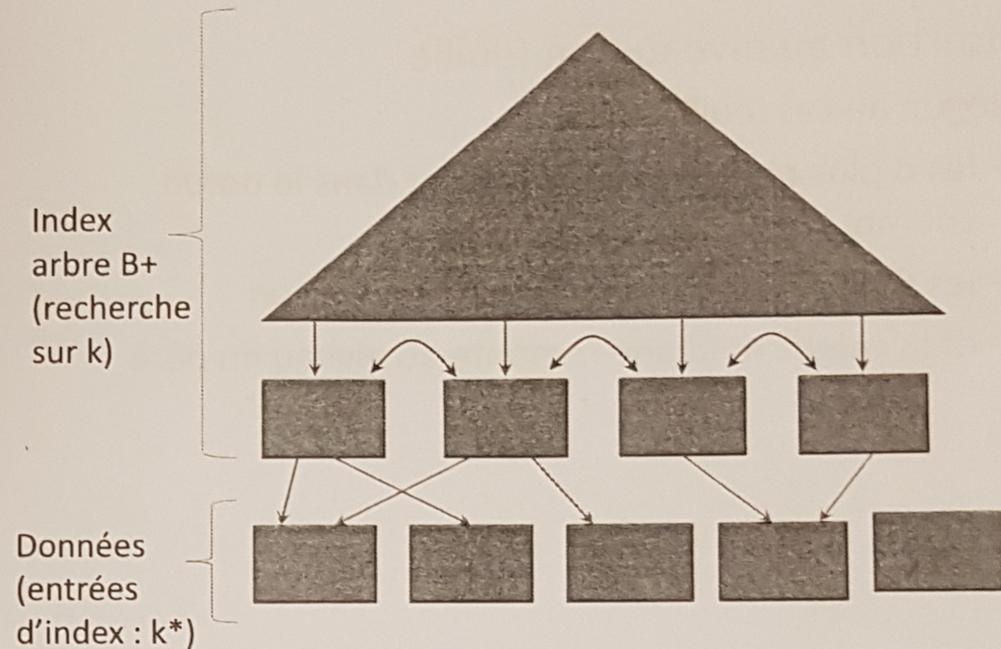
16

Généralités sur les arbres B

*d=2
généralment*

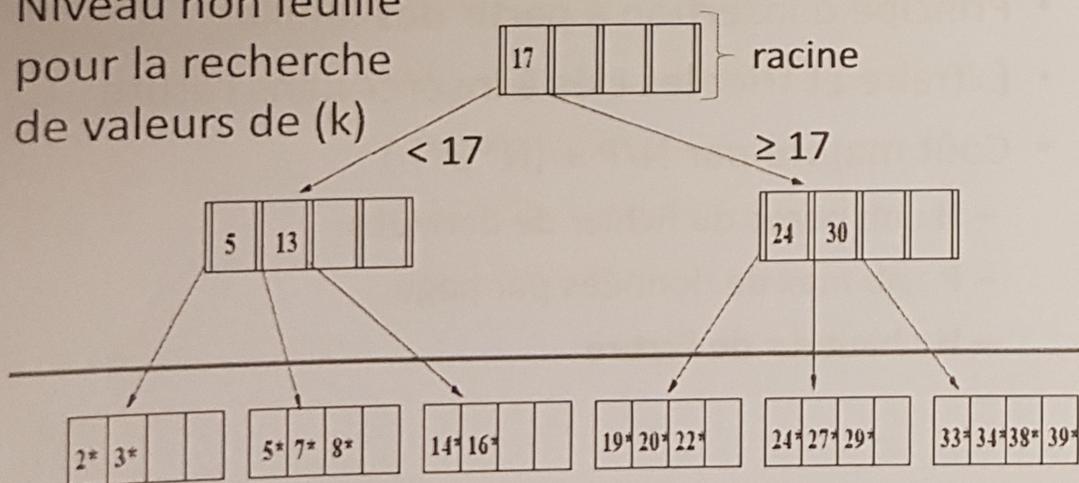
- Arbre B d'ordre d
 - Chaque nœud contient k clés triées avec
 - $d \leq k \leq 2d$ pour les nœuds non racines
 - $1 \leq k \leq 2d$ pour la racine
 - Chaque chemin de la racine à une feuille a la même hauteur
 - Un nœud est
 - terminal (feuille)
 - Ou bien il a k+1 fils et les clés du i^{ème} fils ont des valeurs comprises entre les valeurs des i-1^{ème} et j^{ème} clés du père

Illustration



Exemple

Niveau non feuille pour la recherche de valeurs de (k)



Niveau feuille (k*)

C'est au niveau feuille que l'on accède aux données (elles-mêmes ou à un pointeur vers les données selon le type d'index 1, 2 ou 3)

Principe d'insertion (arbre B) 1/2

- Insertion dans une feuille
- Recherche de la feuille d'insertion
- Si la feuille n'est pas pleine alors l'insérer à sa place
- Sinon (la feuille a déjà 2d clés)
 - Laisser les d+1 plus petites valeurs dans le nœud
 - Créer un nouveau nœud et y placer les d plus grandes valeurs
 - Remonter la plus petite clé du nouveau nœud dans le nœud père
 - Relancer récursivement

Principe d'insertion (arbre B) 2/2

- Insertion au niveau non feuille

Analogie au cas feuille **mais**

- les **d** plus petites valeurs restent dans le nœud courant
- les **d** plus grandes dans le nouveau nœud
- et la valeur médiane remonte au niveau du père

21

Principe de suppression (1/2)

Suppression dans une feuille

Si nb clés restantes $\in [d, 2d]$

→ Supprimer et tasser

Si nb clés restantes $< d$

cas 1 : Si 1 voisin a strictement plus de d clés alors

→ Prendre une clé à voisin

→ Mettre à jour les clés du père *On remplace la valeur du père de tout nœud qui a une nouvelle valeur de gauche*

cas 2 : Si voisin a un nb $\leq d$ clés

→ prendre toutes ses valeurs et supprimer voisin

→ supprimer la clé du père inutilisée

remplir tjs le nœud le plus à gauche

[si on fusionne avec le voisin de gauche : la fusion se fait dans le voisin sinon dans le nœud courant]

22

Principe de suppression (2/2)

Suppression dans un nœud non feuille

Si nb clés restantes $\in [d, 2d]$

→ Supprimer et tasser

Si nb clés restantes $< d$

Redistribuer les données uniformément en passant par le nœud père

23

Bulk loading : Création d'un arbre B

- Principe d'insertion à partir des feuilles
- Extraire et trier les clés à insérer dans l'arbre
- Coût majoré par $N/P + (N \cdot 3 \cdot h)$
 - N : nb pages du fichier de données
 - P : nb max de données par page
 - H : hauteur de l'arbre

24

Et les doublons ?

- Lorsqu'une même valeur de clé d'index est présente plusieurs fois, on crée des *pages de débordement*
 - Stocker les pointeurs vers tous les enregistrements des personnes qui ont 18 ans
- Deux stratégies possibles (dans un arbre)
 - Débordement par valeur de clé
 - A chaque nouvelle valeur déjà dans l'arbre, on crée une feuille de débordement dédiée à cette valeur
 - Une feuille pour les personnes de 18 ans, une pour 19, une pour 20...
 - Débordement par feuille
 - Toutes les valeurs qui apparaissent au moins deux fois sont insérés ensemble dans des feuilles

nombre 1000
20 slots per entrée index
10000 page (Table)
page = 20 n - vplats
500 page