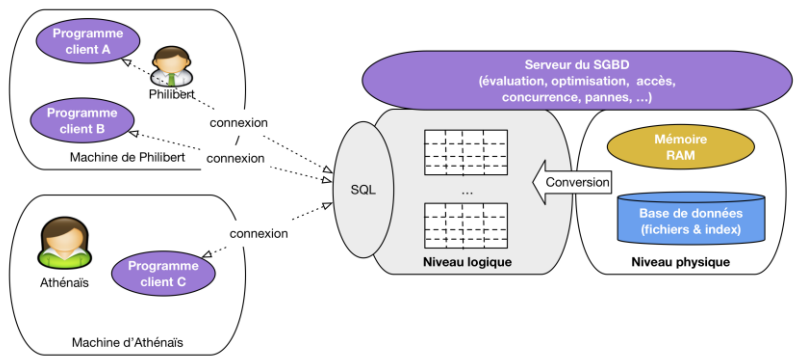
Systèmes de Bases de données relationnelles

http://www.bdpedia.fr/

C.J Date Inroduction aux bases de données Vuibert 2001

Extrait du cours de P. Rigaux



Plan

- **Organisation physique**
- Optimisation
- Concurrence

Une organisation très simple
(Extrait du cours de Jacques Le Maitre)

- Soit la BD relationnelle :
  - `livre(titre, auteur)`
  - `personne(nom, prénom, âge)`
- La relation `livre` est stockée dans un fichier `livre.txt` dont chaque enregistrement représente un doublet de la relation `livre` et a 2 champs : les valeurs des attributs `titre` et `auteur` de ce triplet.
- La relation `personne` est stockée dans un fichier `personne.txt` dont chaque enregistrement représente un triplet de la relation `personne` et a 3 champs : les valeurs des attributs `nom`, `prénom` et `âge` de ce triplet.
- Deux métarelations décrivant les relations de la BD et leurs attributs :
  - `relation(nom, nb_att)`
  - `attribut(nom_table, nom, type, rang)`

  sont elles-même stockées dans les fichiers `relation.txt` et `attribut.txt`.

La BD et les fichiers

livre	
titre	**auteur**
BD et SGBD	Dupont
XML	Durand

personne		
nom	**prénom**	**age**
Dupont	Jean	18
Durand	Pierre	20

fichier `table.txt`
personne|3
livre|3
fichier `attribut.txt`
livre|titre|texte|1
livre|auteur|texte|2
personne|nom|texte|1
personne|texte|2
personne|age|entier|3
fichier `livre.txt`
BD et SGBD|Dupont
XML|Durand
fichier `personne.txt`
Dupont|Jean|18
Durand|Pierre|20

Evaluation de requête

La requete :

```
SELECT livre.titre
FROM livre, personne
WHERE livre.auteur = personne.nom AND
      personne.age = 30;
```

L'algorithme :

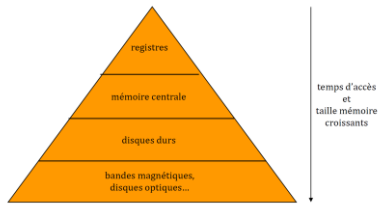
- **pour chaque** enregistrement *l* du fichier `livre.txt`
  pour chaque enregistrement *p* du fichier `personne.txt`
  si *l*.3 = *p*.1 et *p*.3 = 30 **alors** afficher *l*.2;

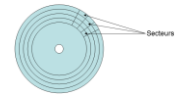
Objectif

- Une BDR est constituée d'un ensemble de relations qui ont chacune une extension qui est un ensemble de n-uplets
- Ces n-uplets sont physiquement stockés dans un ou plusieurs fichiers qui peuvent être répartis sur un ou plusieurs sites (BD distribuées)
  - → Un SGF est au cœur d'une BD
- Le format de stockage choisi doit permettre
  - une utilisation optimale de la mémoire,
  - un accès rapide et des mises à jour peu coûteuses

Fichier

- Les données sont stockées dans des pages
- Une page est stochée dans un bloc d'un disque
- Un bloc est stocké sur plusieurs secteurs consécutifs du disque
- Un fichier occupe un ou plusieurs blocs (pages) sur un disque
- L'accès aux fichiers est géré par un logiciel spécifique
  - le Système de Gestion de Fichiers (SGF)





Rappels

- *Fichier*
  - un récipient de données identifié par un nom et contenant des informations système ou utilisateur
- *Article* (Record) – *Enregistrement* – *n-uplet*
  - Elément d'un fichier, correspond à l'unité de traitement par les programmes d'application
- Caractéristiques d'un fichier

  NOM - CREATEUR - DATE DE CREATION –
  UN OU PLUSIEURS TYPES D'ENREGISTREMENT - UN EMPLACEMENT EN MS
  UNE ORGANISATION

Organisation de fichier

- Mode de répartition des articles dans les pages
  - nature des liaisons entre les articles contenus dans un fichier

- Voici trois sortes d'organisation principales :
  - Fichiers séquentiels
  - Fichiers indexés (séquentiels indexés et arbres-B)
  - Fichiers hachés ou aléatoires

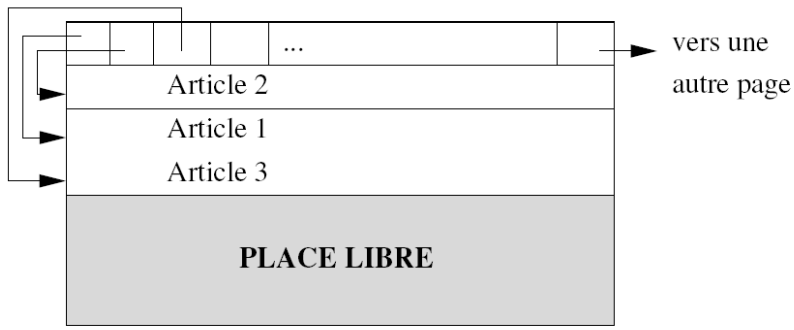
Opérations principales sur un fichier

- Ouvrir / fermer le fichier
- Insérer un n-uplet
- Modifier un n-uplet
- Détruire un n-uplet
- Rechercher un ou plusieurs n-uplets

- Hypothèse
  - Le coût d'une opération est surtout fonction du nombre d'E/S (nb de pages échangées)

Graphique tiré du cours SGBD cnam Equipe vertigo
http://deptinfo.cnam.fr/new/spip.php?article685

Structure interne d'une page



L'adresse d'un n-uplet est constituée de
- L'adresse de la page dans laquelle il se trouve
- Un entier : indice dans un répertoire placé en début de page qui contient l'adresse réelle du n-uplet dans la page

N-uplet / article / enregistrement

- Un n-uplet est une séquence de champs (attributs)
  - en format fixe
    - les valeurs d'attributs sont enregistrées dans des champs de longueur fixe
  - en format variable
    - elles sont stockées les unes derrière les autres précédées de leur longueur, ou de leur nom
- Les n-uplets sont stockés dans les pages
  - On suppose que taille n-uplet < taille de page

Représentation d'un n-uplet

```
CREATE TABLE personne (
    nom CHAR(15),
    prenom CHAR(10),
    age SMALLINT);
```

- Format fixe

`Dupont_____Jean_____30` → 27 octets

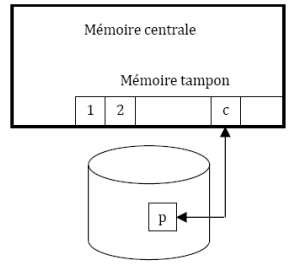
- Format variable

`6Dupont4Jean230` →longueur

`nom#Dupont#prénom#Jean#age#18` →nom

E/S

- Les échanges de données entre disque et mémoire centrale se font au travers de la mémoire tampon (buffer)
- Elle est constituée d'une suite de cases dont chacune peut contenir une page
- En mémoire centrale, une page est repérée par le numéro de la case où elle est rangée dans le tampon



Recherche d'une page

- L'opération de recherche d'une page
  - a pour argument l'adresse *p* de la page cherchée,
  - retourne l'adresse *c* de la case du tampon dans laquelle cette page est rangée
- Principe de l'algorithme
  - Si la page *p* est dans la case *c* du tampon, retourner *c*
    → On économise un accès disque
  - Si la page *p* n'est pas dans le tampon, il faut la lire sur le disque. On teste s'il existe une case libre pour la recevoir
    - Si oui, case *c*
    - Sinon, il faut libérer une case et donc renvoyer une page du tampon sur le disque (FIFO, LIFO, sous contrôle du SGBD)
      - Si la page à rejeter a été modifiée pendant son séjour en mémoire centrale, il faut la réécrire sur le disque →travail du gestionnaire de transactions
      - Transférer la page *p* du disque dans la case *c* et retourner *c*

Stratégies de remplacement de page

- FIFO First-In First-Out
  - On renvoie sur disque la page qui n'a pas été utilisée depuis le plus longtemps.
  - hypothèse : cette page a moins de chances d'être réutilisée que les autres
- LIFO Last-In First-Out
  - On renvoie sur disque la page qui a été utilisée le plus récemment
  - Intérêt : sa simplicité, car n'y a pas besoin de mémoriser les dates auxquelles les pages ont été chargées dans le tampon
- Sous contrôle du SGBD
  - Le SGBD peut punaiser («to pin») des pages dans la mémoire tampon, afin qu'elles ne soient pas renvoyées sur disque, car il sait qu'elles vont être réutilisées.

Temps d'accès à un n-uplet

- L'accès à un *n*-uplet consiste :
  1. à rechercher sur le disque la page qui le contient
  2. à le rechercher dans cette page
- Le temps d'accès à un *n*-uplet est donc égal à :

  $T_{recherche\ dans\ page} + T_{transfert\ page}$

- $T_{recherche\ dans\ page}$ est négligeable par rapport à $T_{transfert\ page}$
- Le temps d'accès à une page dépend de sa localisation (mémoire tampon ou disque)
- En conclusion, il y a intérêt :
  - à regrouper dans une même page les *n*-uplets traités consécutivement,
  - à traiter consécutivement les *n*-uplets d'une même page.

Organisation

- *Organisation de fichier*
  - le mode de répartition des enregistrements dans les pages

- Trois sortes d'organisation principales :
  - Fichiers séquentiels
  - Fichiers indexés (séquentiels indexés et arbres-B)
  - Fichiers hachés ou aléatoires

Organisation séquentielle

- Insertion
  - les articles sont stockés séquentiellement dans les pages au fur et à mesure de leur création
- Recherche
  - le fichier est parcouru séquentiellement
- Destruction
  - recherche, puis destruction (par marquage d'un bit par exemple)
- Modification
  - recherche, puis réécriture

Côut des opérations

- Nombre moyen de lectures/écritures sur disque d'un fichier de n pages
  - Recherche : n /2 - On parcourt en moyenne la moitié du fichier
  - Insertion : n + 1 - On vérifie que l'article n'existe pas avant d' écrire
  - Destruction et mises-à-jour : n/2 + 1

→ organisation utilisée pour les fichiers de petite taille

Fichiers séquentiels triés

- Une première amélioration consiste à trier le fichier sur sa clé d'accès
- On peut alors effectuer une recherche par dichotomie
- Coût de l'opération log2(n)

Ajout d'index

- Un index est un second fichier
- Un index est construit pour accéder de façon sélective, et donc rapide, aux enregistrements d'un fichier de données *D*, dont la valeur *c* d'un champ ou d'une liste de champs est donnée
  - On appelle *c* clé de recherche

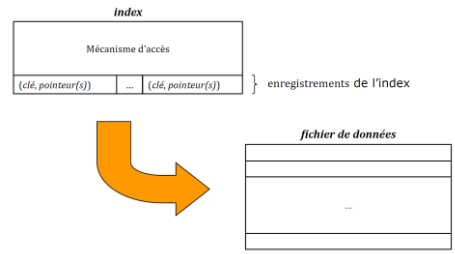
Coût d'une recherche avec et sans index

- Soit un fichier F contenant 1000 pages
- On suppose qu'une page d'index contient 100 entrées, et que l'index occupe donc 10 pages

- F non trié et non indexé.
  - Recherche séquentielle 500 pages.
- F trié et non indexé.
  - Recherche dichotomique : $\log_2(1000) \sim 10$ pages
- F trié et indexé.
  - Recherche dichotomique sur l'index, puis lecture page : $\log_2(10) + 1 \sim 5$ pages

Ajout d'index

- Un index est composé de deux parties
  - un fichier d'enregistrements à 2 champs
    - une clé de recherche *c*
    - un pointeur ou une liste de pointeurs vers des enregistrements de *D*
  - un mécanisme d'accès à un enregistrement de l'index à partir de la clé de recherche



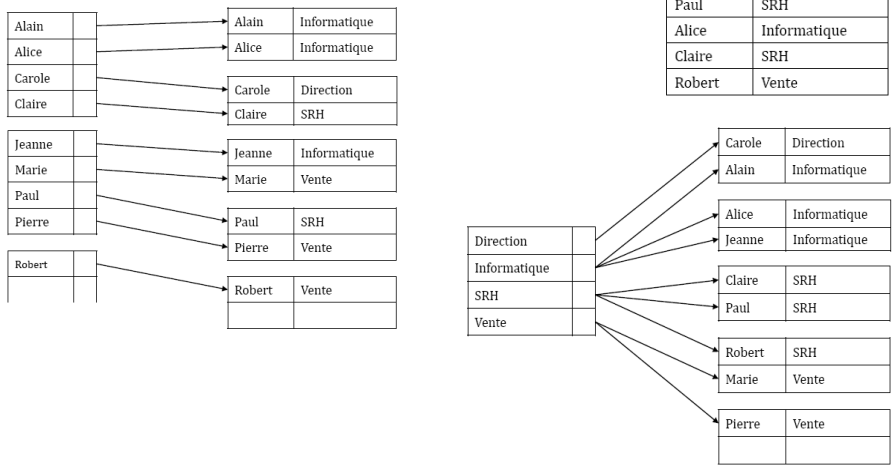
Index primaire - index secondaire

- Un index est primaire si la clé de recherche est une clé d'un enregistrement du fichier de données
  → un enregistrement de l'index pointe vers un seul enregistrement du fichier de données

- Un index est secondaire sinon
  → un enregistrement de l'index peut pointer vers plusieurs enregistrements du fichier de données

Index primaire/secondaire

employé	
nom	**département**
Alain	Informatique
Pierre	Vente
Marie	Vente
Jeanne	Informatique
Carole	Direction
Paul	SRH
Alice	Informatique
Claire	SRH
Robert	Vente

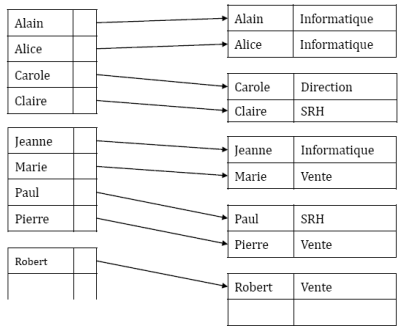


Index dense – Index creu

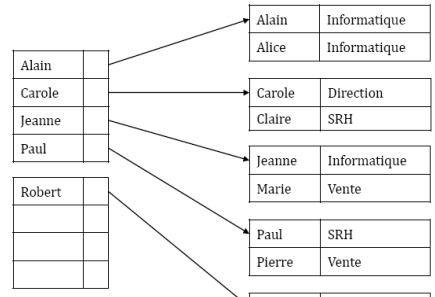
- Un index est dense si chaque clé de recherche dans le fichier de données apparaît dans l'index
- Un index est creux si seulement certaines clés du fichier de données apparaissent dans l'index (en général, une clé par bloc du fichier de données)

Index primaire

employé	
nom	**département**
Alain	Informatique
Pierre	Vente
Marie	Vente
Jeanne	Informatique
Carole	Direction
Paul	SRH
Alice	Informatique
Claire	SRH
Robert	Vente



Index primaire dense



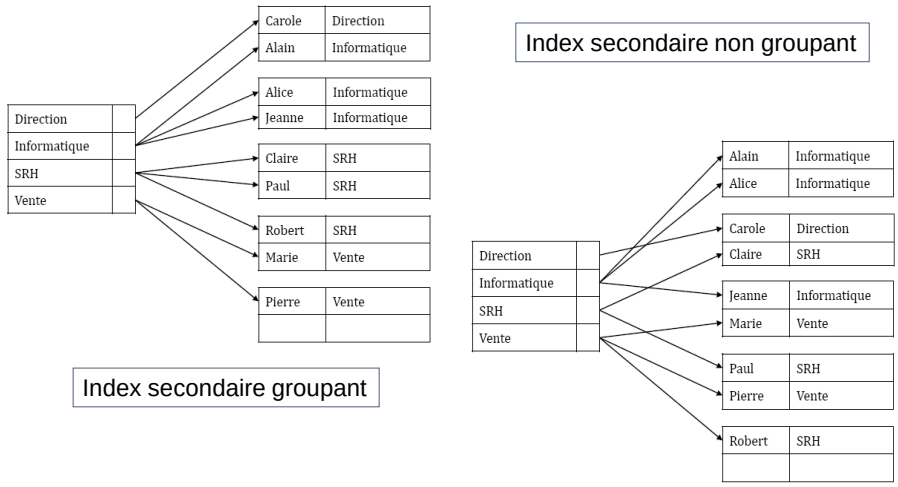
Index primaire creux

Index groupant– Index non groupant

- Un index dense est groupant, si ses enregistrements et ceux du fichier de données sont triés selon la clé de recherche
- Sinon il est non groupant

- Un index secondaire est toujours dense

Index secondaire



Index dans les BDR

- Les index sont construits sur une relation par la commande SQL:
  `CREATE INDEX` *`nom ON relation(liste d'attributs)`*
- Pour une relation *R,* on distingue *:*
  - l'index primaire construit sur la clé primaire de *R*
  - les index secondaires construits sur un attribut (ou une liste d'attributs) de *R*

Mécanismes d'accès aux enregistrements d'un index

- Organisation arborescente
  - Séquentiel indexé
  - Arbres B+
- Accès par hachage
  - Statique
  - Dynamique

- L'ensemble des supports utilisés pour présenter ces mécanismes sont extraits de supports de Jacques Le Maitre

Table exemple sur laquelle l'index primaire sera construit

dictionnaire	
mot	**définition**
mélodie	Suite de sons formant un air...
école	Etablissement où se donne un enseignement collectif...
nez	Partie saillante du visage...
bateau	Nom des embarcations, des navires...
kayak	Embarcation étanche et légère...
zébu	Bœuf à longues cornes et à bosse sur le garrot...
dessin	Représentation sur une surface de la forme d'un objet...
corde	Assemblage de fils tressés ou tordus ensemble...
terre	Planète habitée par l'homme...

Arbre B+

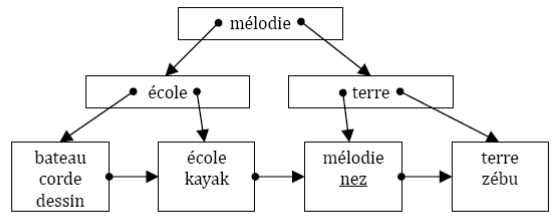
- Index hiérarchisé
  - si l'index est grand, la recherche d'une clé dans l'index peut être longue
  - Créer un index sur l'index etc…
- Arbre B - balanced tree - arbre équilibré
  - une structure arborescente dans laquelle tous les chemins de la racine aux feuilles ont même longueur
  - introduit par Bayer et McCreight en 1972

→ Arbre B+
- variante des arbres B
- très utilisé en BD pour construire des index

Un exemple d'arbre B+

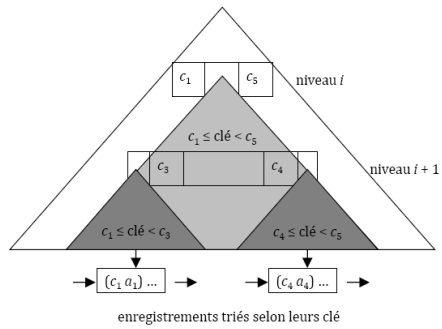
- Ordre du B+ arbre m = 3
- On ne représente que les clés des enregistrements

mot
mélodie
école
nez
bateau
kayak
zébu
dessin
corde
terre



Arbre B+

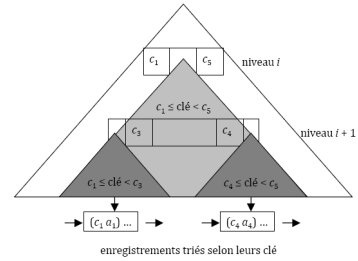
- Un arbre B+ d'ordre *m* (entier impair ≥ 3) est un arbre équilibré dont chaque nœud est enregistré dans une page stockée sur disque



Arbre B+ - les feuilles

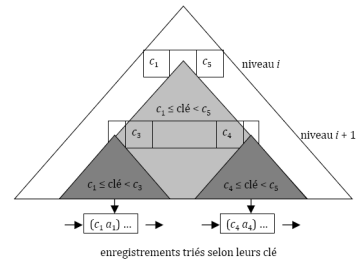
- Une feuille contient une séquence d'enregistrements $(c_1\ a_1)\ \ldots\ (c_k\ a_k)\ p$
  - c_i *est une clé et* a_i *est l'information associée à cette clé*
  - p *est un pointeur vers la feuille suivante*

  triée par ordre croissant de clé
- Une feuille est au moins à moitié remplie $(m+1)/2 \leq k \leq m$ sauf si elle est l'unique nœud de l'index
- Les feuilles sont chaînées entre elles dans l'ordre de leur première clé à l'aide du pointeur p



Arbre B+ - les nœuds non terminaux

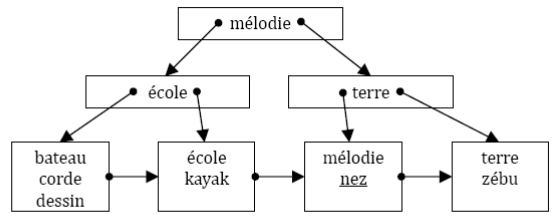
- Un nœud NT contient une séquence d'enregistrements (p_1) … $(c_k\ p_k)$
  - c_i *est une clé*
  - p_i (i compris entre 1 et m-1) est un pointeur vers la racine du sous-arbre dont les feuilles contiennent les enregistrements dont la clé est >= c_i et <c_{i+1}
  - p_m est un pointeur vers la racine du sous-arbre dont les feuilles contiennent les enregistrements dont la clé est >= à c_m
- Un nœud est au moins à moitié rempli
  $(m+1)/2 \leq k \leq m$
  sauf s'il est la racine, auquel cas son contenu peut se réduire à
  $(p_1)\ (c_2\ p_2)$



L'index exemple en arbre B+

- Ordre du B+ arbre m = 3
- On ne représente que les clés des enregistrements

mot
mélodie
école
nez
bateau
kayak
zébu
dessin
corde
terre



Recherche d'un enregistrement

(Il s'agit de rechercher l'enregistrement dont la clé c est donnée ou, s'il n'existe pas, le nœud qui devrait le contenir.)

début

Le nœud courant est la racine de l'arbre B+.

tant que le nœud courant est un nœud non terminal de contenu (p_1) (c_2, p_2)... (c_n, p_n) **répéter**

si $c < c_2$ **alors**

Accéder au nœud d'adresse p_1 qui devient le noeud courant.

sinon

Rechercher séquentiellement le dernier c_i inférieur ou égal à c et accéder au nœud d'adresse p_i qui devient le nœud courant.

fin

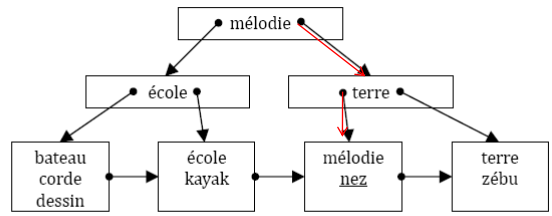
fin

Rechercher l'enregistrement de clé c dans le nœud courant (une feuille).

fin

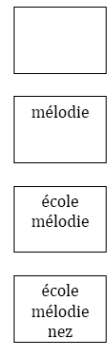
L'index exemple en arbre B+

- On recherche `nez`

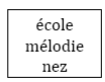


Insertion d'un n-uplet
Construction de l'index pas à pas

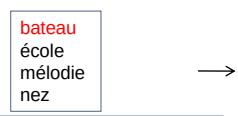
Insertions successives des enregistrements : mélodie, école, nez.



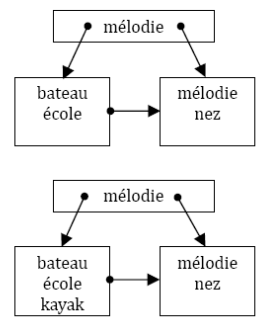
Construction de l'index pas à pas



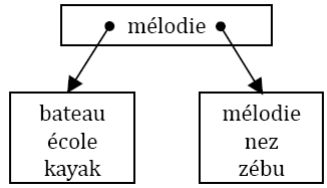
Insertions successives des enregistrements : bateau, kayak.



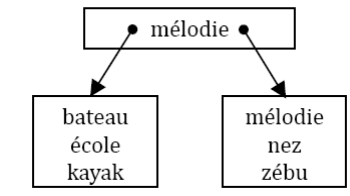
Bateau → Fission de nœud



Insertion de zébu



Insertion de dessin



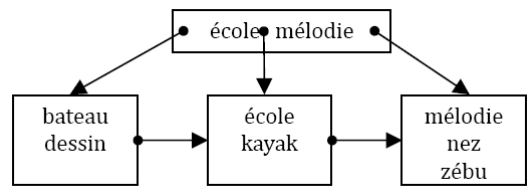
bateau
dessin
école
kayak

Dessin → Fission de nœud

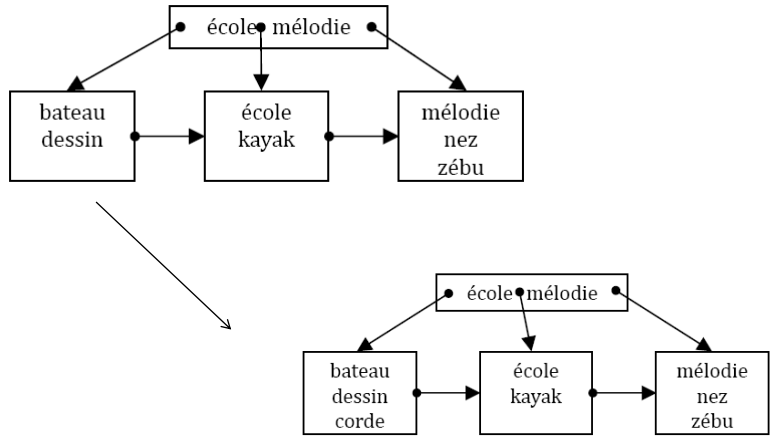
bateau
dessin

école
kayak

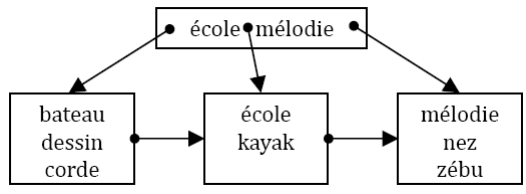
On remonte la clé médiane



Insertion de corde



Insertion de terre



mélodie
nez
terre
zebu

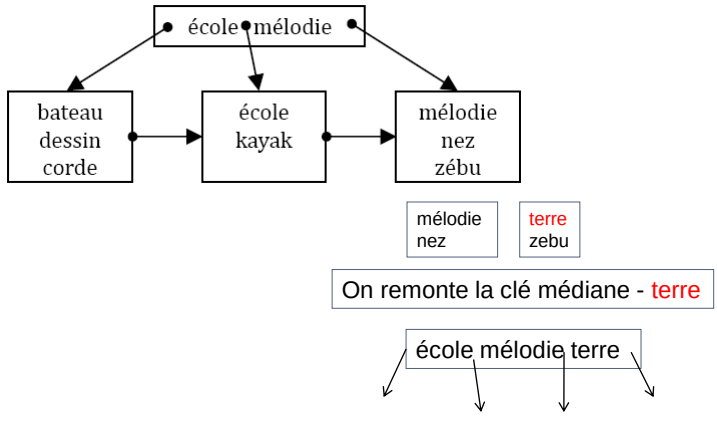
Terre → Fission de nœud

mélodie
nez

terre
zebu

On remonte la clé médiane - terre

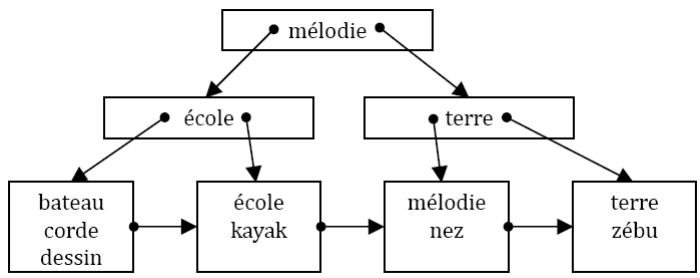
Insertion de terre



NT trop plein → fission du nœud
4 pointeurs (donc 4 clés) or m=3)

Insertion de terre

On crée deux nds NT, on remonte la clé médiane
mais on la supprime au niveau où la fission s'est faite



Insertion d'un enregistrement

(Il s'agit d'insérer l'enregistrement de clé c et d'information associée a.)
début
  Rechercher l'enregistrement de clé c.
  si il existe **alors**
    L'insertion est terminée.
  sinon
    Le nœud courant est celui sur lequel s'est arrêtée la recherche
    et l'enregistrement à insérer est (c, a).
    répéter
      Soit n le nœud courant, p son adresse et s le contenu de n.
      Insérer l'enregistrement à insérer dans s en respectant l'ordre des clés.
      si *longueur*$(s) \leq m$ **alors**
        Enregistrer s dans n.
        L'insertion est terminée.
      sinon
        fission du nœud courant
      fsi
    jusqu'à ce que l'insertion soit terminée
  fsi
fin
(La longueur d'une séquence est le nombre d'enregistrements qui la composent.)

Fission d'un nœud

(Il s'agit de répartir sur deux nœuds le nouveau contenu s du nœud n d'adresse p)
début
  Créer un nouveau nœud n' d'adresse p'.
  Découper s en deux séquences s_1 et s_2 de longueur égale :
  c'est possible, car la longueur de s est $m + 1$ qui est un nombre pair.
  Soit c_{21} la première clé de s_2.
  Enregistrer s_1 dans n.
  Enregistrer s_2 dans n' après avoir supprimé sa première clé,
  si n' est un nœud non terminal.
  si n est une feuille **alors**
    Chaîner n' à n.
  fin
  si n est la racine **alors**
    Créer un nouveau nœud de contenu (p) $(c_{21}\ p')$.
    L'insertion est terminée (la hauteur de l'index a augmenté de 1).
  sinon
    Le père de n devient le nœud courant
    et $(c_{21}\ p')$ devient l'enregistrement à insérer.
  fsi
fin

Performances

- Le nombre de lectures de nœuds pour accéder à un enregistrement est égal à la longueur d'une branche → la hauteur de l'arbre
- La hauteur maximum (*hmax*) de l'arbre est obtenue quand la racine est réduite à deux enregistrements et les autres nœuds ne sont remplis qu'à moitié, c.-à-d. ne contiennent que $(m + 1) / 2$ enregistrements

- Calcul de *hmax* en posant $n = (m+ 1) / 2$
  - au 1er niveau, l'arbre possède 2 enregistrements
  - au 2e niveau, il en possède 2n
  - au 3e niveau, il en possède $2n^{3-1}$
  - au ième niveau, il en possède $2n^{i-1}$

- Soit *N* le nombre d'enregistrements de l'index
  On a
  $2n^{hmax-1} = N$ et donc $hmax = log_n(N/ 2) + 1$

Performances

- Par exemple, si l'on suppose
  - que l'on peut ranger 99 enregistrements par nœud,
  - qu'il y a 10^6 enregistrements dans l'index

alors $m= 99, n= 50$

- On a $\log_{50}(10^6/2) + 1$ soit 3,85

→ Il faut donc 4 lectures de bloc disque dans le pire cas pour retrouver un enregistrement à partir de sa clé

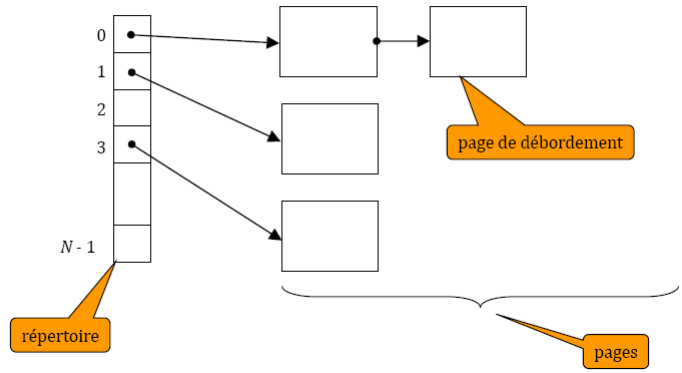
Mécanismes d'accès aux enregistrements d'un index

- Organisation arborescente
  - Séquentiel indexé
  - Arbres B+
- Accès par hachage
  - Statique
  - Dynamique

Hachage

- Répartir aléatoirement les n-uplets dans des paquets composés d'une ou plusieurs pages en fonction de leur clé
- On utilise une fonction de hachage
  - qui s'applique à la clé d'un nuplet
  - et fournit l'adresse de ce nuplet
- Ceux dont la valeur est la même sont dans le même paquet
- Le hachage est *statique* ou *dynamique* selon la fonction de hachage est fixée ou évolue durant la vie de l'index
- Un index à accès par hachage peut être organisé avec ou sans répertoire

Hachage statique avec répertoire : organisation



Obtenir une distribution uniforme pour éviter de saturer un paquet
AUTORISER LES DEBORDEMENTS

Hachage statique : organisation

Un index à accès par hachage statique avec répertoire est un quadruplet (N, h, P, R) où :

- N est un nombre entier positif,
- h est une **fonction de hachage** qui appliquée à une clé produit un nombre entier compris entre 0 et N - 1, appelé **code haché** (« hash-code »).
- P est un ensemble de pages stockées sur disque
  Chaque page contient une liste d'enregistrements $(c\ a)$ où :
  - c est la clé,
  - a est l'information associée.
- R est un répertoire de N cases.
  Chaque case i $(0 \leq i \leq N - 1)$ est le début d'une chaîne (éventuellement vide) de pages dont tous les enregistrements sont tels que $H(c) = i$.
- Les pages de cette chaîne à partir de la 2ème position sont appelées pages de **débordement** (« overflow »).

Fonction de hachage

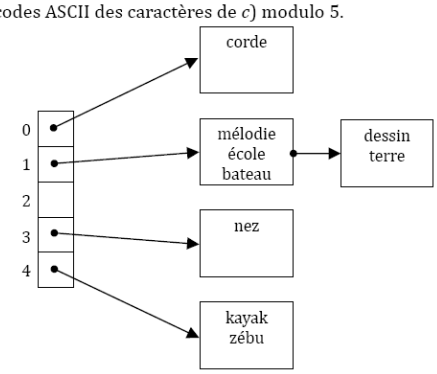
- La fonction de hachage
  - s'applique à une clé - une chaîne de caractères - et à un nombre entier N
  - retourne un nombre entier compris entre 0 et *N-1*
- Il n'est en général pas possible de construire une fonction de hachage injective
  c1≠ c2 => h(c1) ≠ h(c2)
  → On construit donc une fonction qui minimise le nombre de collisions et les répartit uniformément

- Deux techniques très utilisées : Division et pliage
- On construit à partir des codes des caractères de la clé un nombre k grand devant N
  - Division
    h(c) est le reste de la division de *k* par *N (*avec *N* premier)
  - Pliage
    On découpe la représentation binaire de *k* en tranches de *b* bits. *h(c)* est égal au «ou exclusif» des nombres binaires ainsi obtenus

Exemple

$N = 5$: répertoire de 5 cases,
3 enregistrements au maximum,
$h(c)$ = (somme des codes ASCII des caractères de c) modulo 5.

c	h(c)
mélodie	1
école	1
nez	3
bateau	1
kayak	4
zébu	4
dessin	1
corde	0
terre	1



Recherche d'un enregistrement

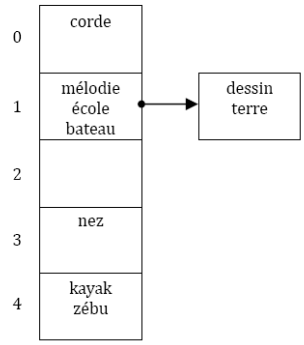
(Il s'agit de rechercher l'enregistrement dont la clé c est donnée.)
début
Parcourir les pages liées à la case $h(c)$
jusqu'à trouver une page qui contienne un enregistrement de clé c :
c'est l'enregistrement recherché.
si la fin de la chaîne est atteinte **alors**
L'enregistrement recherché n'existe pas.
fsi
fin

Insertion d'un enregistrement

(Il s'agit d'insérer un nouvel enregistrement de clé c et d'information associée a.)
début
Rechercher l'enregistrement de clé c.
si il existe **alors**
L'insertion est terminée.
sinon
Parcourir les pages liées à la case $h(c)$ jusqu'à en trouver une qui posséde une place suffisante pour le nouvel enregistrement.
si il en existe une **alors**
Y insérer l'enregistrement : l'insertion est terminée.
sinon (le bout de la chaîne est atteint)
Créer une nouvelle page.
L'ajouter au bout de la chaîne des pages liées à la case $h(c)$.
Y insérer l'enregistrement : l'insertion est terminée.
fsi
fsi
fin

Exemple sans répertoire

On peut éviter l'utilisation d'un répertoire en créant un index de *N* pages contiguës.
Le code haché donne alors directement accès à la page contenant les clés recherchées.



Performances

- Il ne faut pas sous évaluer la valeur de N
- Le hachage statique est mal adapté à un fichier de données très évolutif car la taille du répertoire peut s'avérer sous-évaluée, entraînant un accroissement du nombre de pages de débordement et donc du nombre d'accès disque
- Solution idéale: réorganisation progressive
  - changer la fonction d'adressage ?

Hachage dynamique

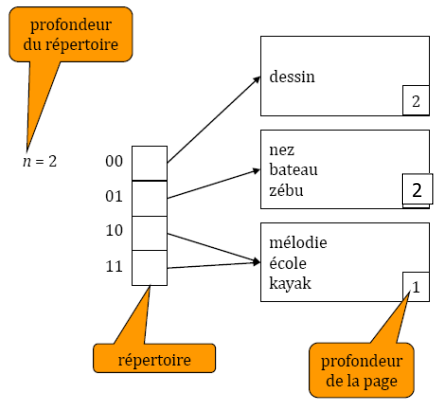
- Si page est saturée, on fait évoluer la fonction de hachage au lieu de créer une page de débordement
  - → assurer que le recherche d'un enregistrement ne nécessitera qu'un seul accès à une page
- Deux méthodes
  - hachage **extensible** - Fagin et al. en 1979
  - hachage linéaire - W. Litwin en 1980

Hachage extensible

- Une suite de bits est associée à chaque clé
- L'évolution de la clé de hachage
  - augmenter le nombre de bits à prendre en compte dans une clé pour trouver la case du répertoire qui pointe vers la page contenant l'enregistrement associé à cette clé
- Si tous les enregistrements tiennent dans une seule page, 0 bits seront à prendre en compte
- S'ils occupent 2 pages, 1 bit sera à prendre en compte,
- S'ils occupent 3 ou 4 pages, 2 bits seront à prendre en compte
- ...

Exemple

c	h(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100

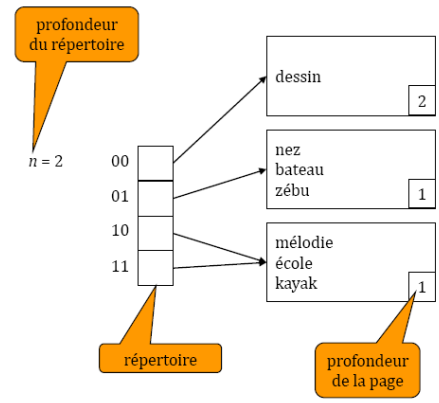


Organisation

- La fonction de hachage h associe à chaque clé une séquence suffisamment longue de bits (32, par exemple).
- L'index est composé de deux parties :
  - un ensemble de pages.
  - un répertoire de 2^n cases ($n \geq 0$, profondeur du répertoire) dont chacune contient un pointeur vers une page.
- Plusieurs cases consécutives du répertoire peuvent pointer vers la même page.
- Un enregistrement de clé c est stocké dans la page pointée par la i^e case du répertoire telle que i est égal au nombre formé par les n premiers bits de $h(c)$.
- A chaque page est associé un nombre entier m ($0 \leq m \leq n$), appelée profondeur de la page.
- Si une page a la profondeur m, alors il y a 2^{n-m} cases du répertoire qui pointent vers elle.

Exemple

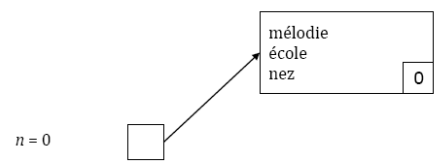
c	h(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100



Construction de l'index

Insertions successives des enregistrements : mélodie, école, nez

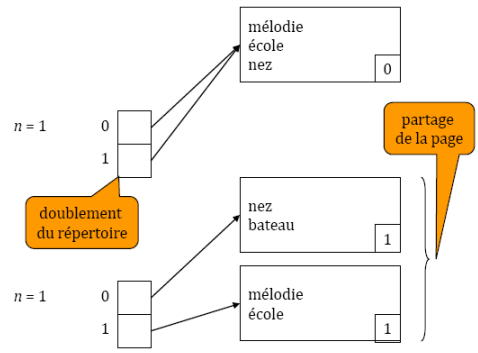
c	h(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101



Construction de l'index

Insertion de l'enregistrement : bateau

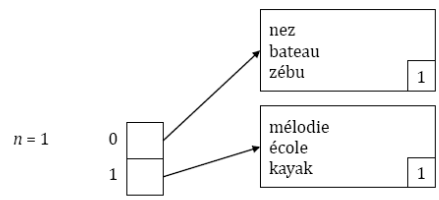
c	h(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101



Construction de l'index

Insertions successives des enregistrements : kayak, zébu

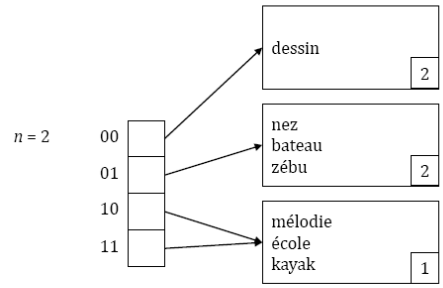
c	h(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101



Construction de l'index

Insertions successives des enregistrements : dessin

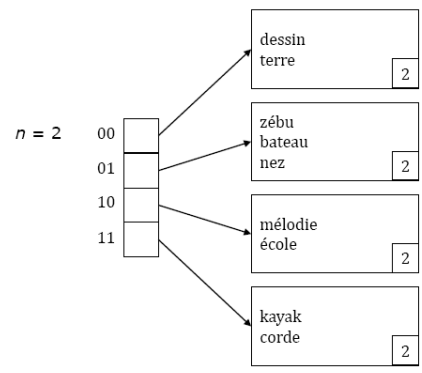
c	h(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101



Construction de l'index

Insertions successives des enregistrements : corde, terre

c	h(c)
mélodie	10110
école	10100
nez	01110
bateau	01000
kayak	11010
zébu	01011
dessin	00100
corde	11000
terre	00101



```
(Il s'agit d'insérer un enregistrement e de clé c)
début
    Rechercher la page P qui devrait contenir e : soit m sa profondeur.
    si P contient e alors
        L'insertion est terminée.
    sinon
        si P n'est pas saturée alors
            Insérer e dans P : l'insertion est terminée.
        sinon
            si m < n alors (partage d'une page en 2)
                Créer une nouvelle page P'.
                m = m + 1
                profondeur locale de P = profondeur locale de P' = m.
                Enregistrer dans P' tous les enregistrements de P
                dont le m^e bit est égal à 1.
                Faire pointer vers P' chaque case du répertoire qui pointait vers P,
                si son m^e bit est égal à 1.
                Recommencer l'insertion de e.
            sinon (doublement du répertoire)
                Doubler la profondeur du répertoire
                en dédoublant chaque case du répertoire.
                n = n + 1.
                Recommencer l'insertion de e.
            fsi
        fsi
    fsi
fin
```

Insertion d'un enregistrement

Avantages - Limites

- Avantages
  - Très rapide : une E/S dans le meilleur des cas pour une recherche
  - Le Hachage n'occupe pas de place disque
- Limites
  - Réorganisation quand le fichier évolue
  - Les recherches par intervalle sont impossibles