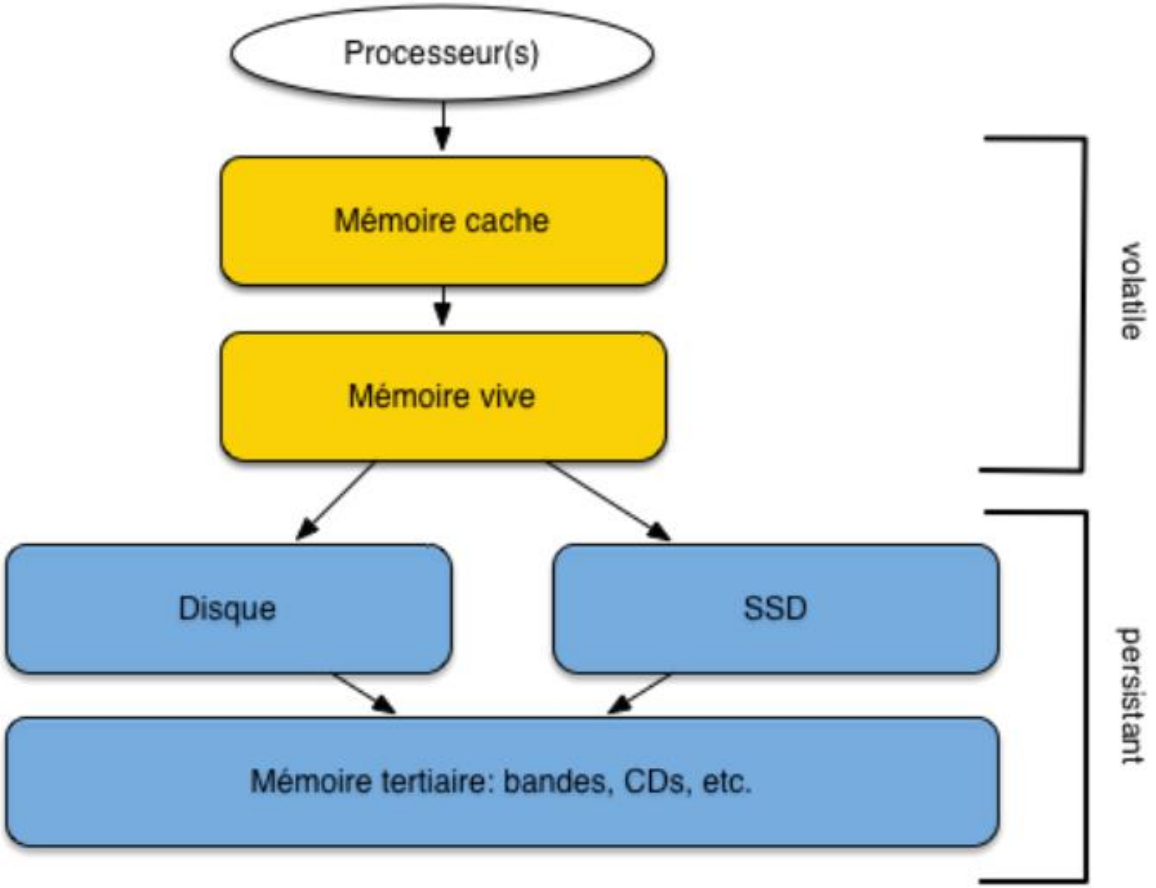
CHAPITRE 7:

STOCKAGE AND INDEXATION

Introduction

- L’organisation des données sur un disque, les structures d’indexation et les algorithmes de recherche utilisés par le SGBD influence les performances.
- Le but principal est de limiter le nombre et la taille de données lues sur le disque.
- Dans ce cours:
  - Le stockage de données sur le disque
  - Les structure d’index et les algorithmees d’accès

Supports de stockage



Source:http://sys.bdpedia.fr/stock.html

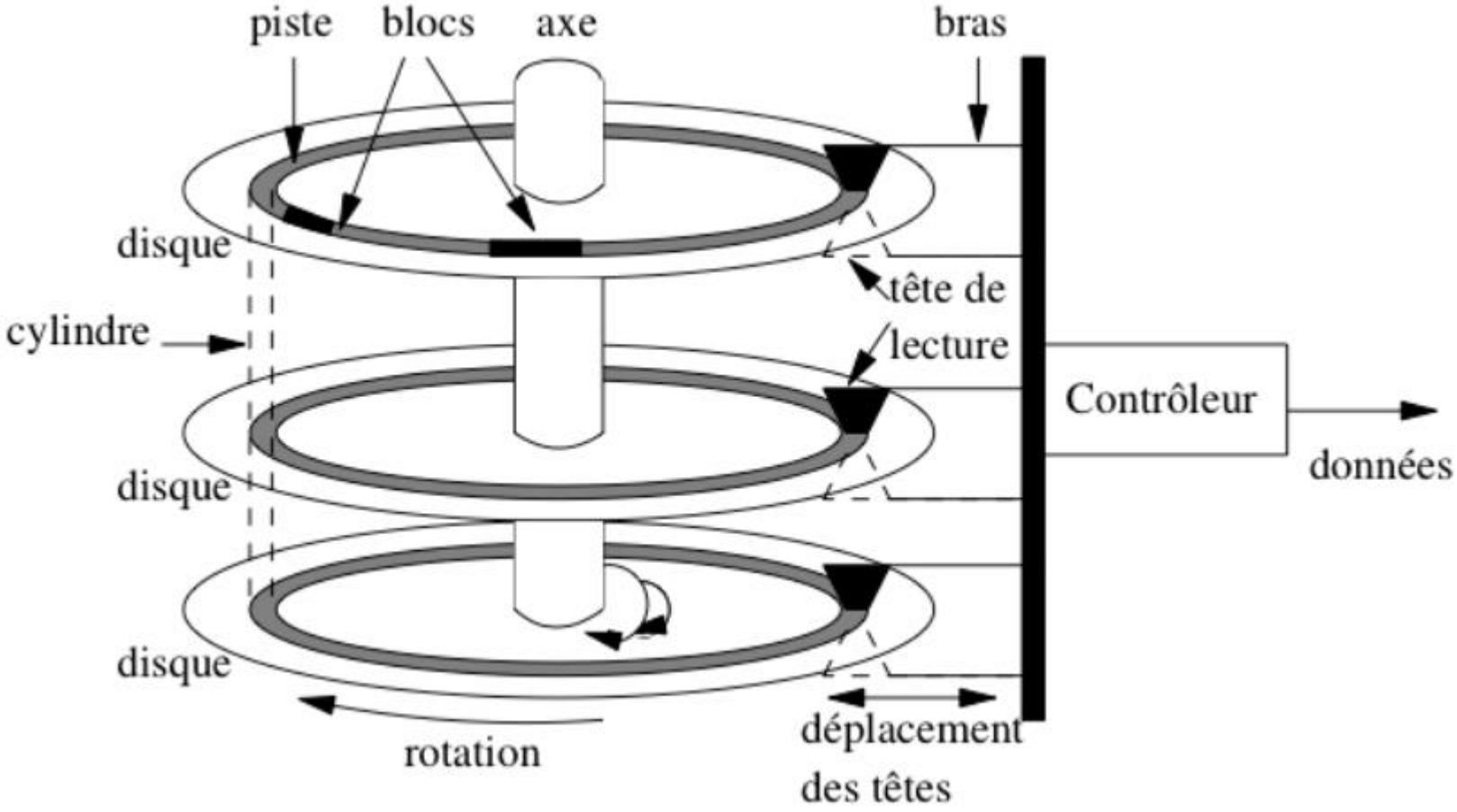
Performance des Supports de stockage

- **Temps d'accès :** connaissant l'adresse d'une donnée, quel est le temps nécessaire pour aller la chercher à l'emplacement mémoire indiqué par cette adresse ?
- **Débit :** volume de données lues par unité de temps dans le meilleur des cas.

Mémoire	Taille	Temps d'accès	Débit
cache	Qq Mo	$\approx 10^{-8}$ (10 nanosec.)	10+ Go/s
Principale (RAM)	Qq Go	$\approx 10^{-8} - 10^{-7}$ (10-100 nanosec.)	Qq Go/s
Disque	Qq To	$\approx 10^{-2}$ (10 millisec.)	100 Mo/s
SSD	Qq To	$\approx 10^{-4}$ (0,1 millisec.)	1+ Go/s

Source:http://sys.bdpedia.fr/stock.html

Stockage sur disque



Fonctionnement d'un disque magnétique.

Source:http://sys.bdpedia.fr/stock.html

Spécification d'un disque: exemple

Caractéristique	Performance
Capacité	2,7 To
Taux de transfert	100 Mo/s
Cache	3 Mo
Nbre de disques	3
Nbre de têtes	6
Nombre de cylindres	15 300
Vitesse de rotation	10 000 rpm (rotations par minute)
Délai de latence	En moyenne 3 ms
Temps de positionnement moyen	5,2 ms
Déplacement de piste à piste	0,6 ms

Source:http://sys.bdpedia.fr/stock.html

Temps de lecture sur disque

- Dans une situation moyenne, la tête n'est pas sur la bonne piste, et une fois la tête positionnée (temps moyen 5.2 ms), il faut attendre une rotation partielle pour obtenir le bloc (temps moyen 3 ms).
- Le temps de lecture est alors en moyenne de **8.2** ms, si on ignore le temps de transfert.

Méthodes de stockage

- Les données d'une BDD (Relations) sont stockées dans la Mémoire Secondaire (MS) (disque).
- La MS est décomposée en segment. Un segment contient plusieurs pages. Une page est un ensemble de blocs.
- La **page** est l'unité de transfert entre la MS et la MC.
- Les données sont chargées en MC sur demande.

Méthodes de stockage

- Les enregistrements sont placés l'un après l'autre dans des pages successives.
- Un enregistrement n'est pas à cheval sur deux pages.
- **L'accès** aux données dans le cas de la recherche d'un enregistrement se fait par un balayage séquentiel des enregistrements.
- **L'insertion** d'un nouvel enregistrement s'effectue dans la dernière page.

Si cette dernière est saturée, alors une nouvelle page est allouée et l'enregistrement est inséré.

- **La suppression** est assurée logiquement grâce à un indicateur de suppression.

Indexation

- L'index est une structure de données qui permet de faire un **accès direct aux données** sur disque.
- L'index est chargé en MC, la recherche sur l'index se fait en MC puis un accès direct se fait sur le disque.
- Seul les données indexées sont chargées en MC.

  => réduction de temps d'accès à la données +

  réduction de temps d'échange entre MC et MS

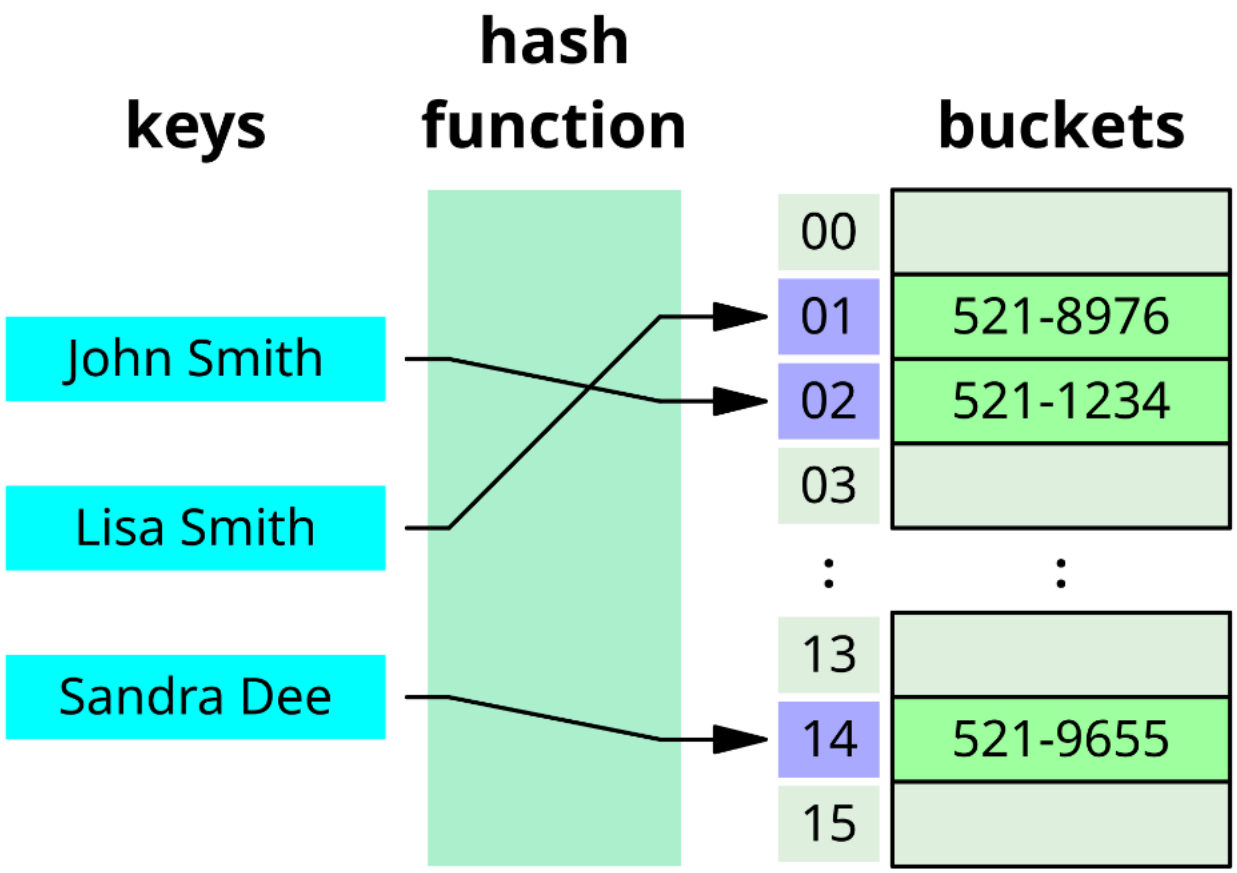
Types d’indexation

- **Hachage**
- **Fichier index**
- **B-arbre**
- **Index bitmap**

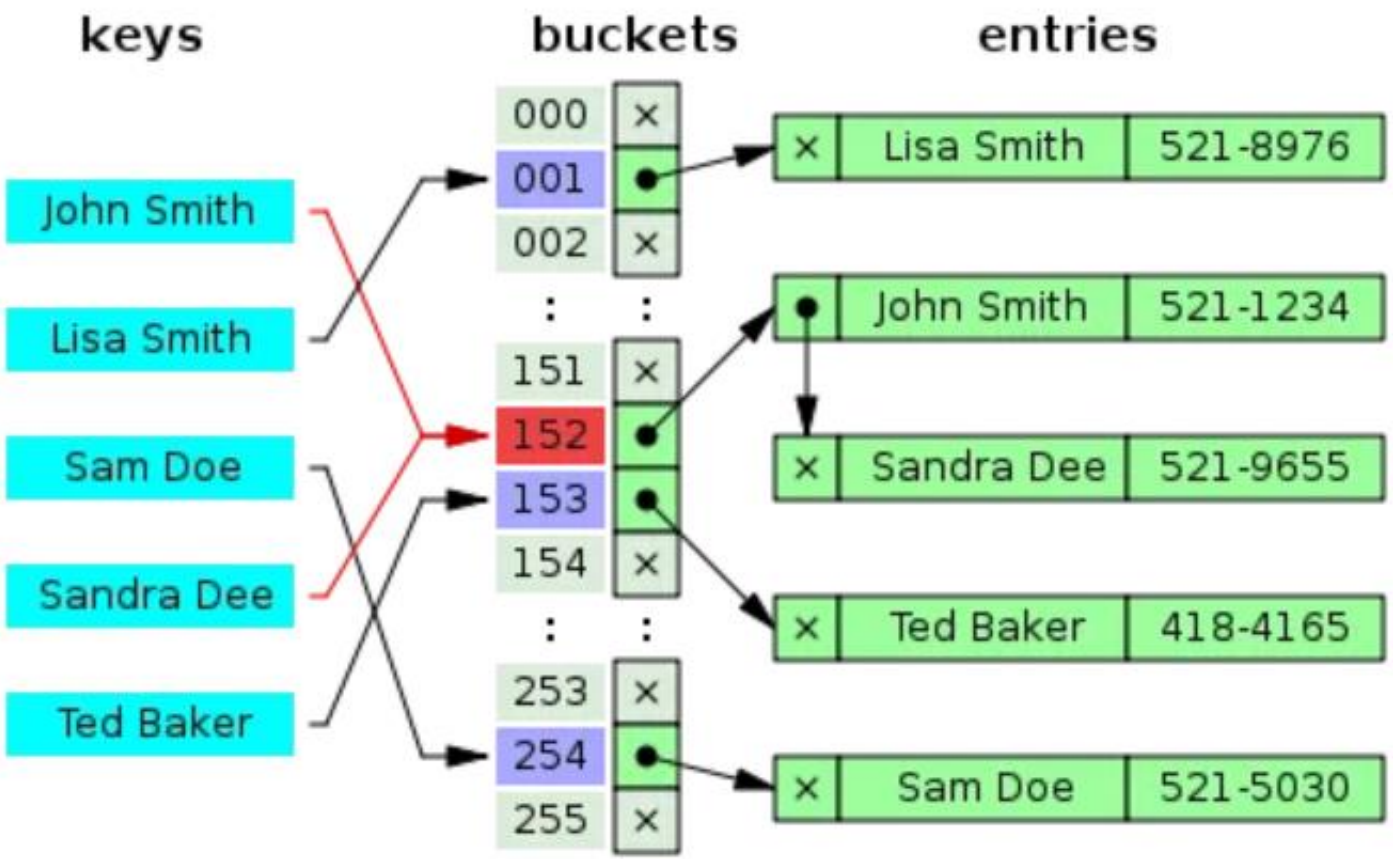
Hachage

- Le stockage est organisé en N fragments **(buckets)** constitués de séquences de blocs. .
- Utilise une **fonction de hachage** pour transformer des clés ou des valeurs en un **code unique**, appelé **hach**.
- Les résultats des fonctions de hachage sont stockés dans une **table de hachage**, où chaque **entrée est liée à un hach** et contient un pointeur vers des **buckets**.

Table de Hachage



En cas de collision du hash



HACHAGE

Algorithme de recherche

- Entrée : valeur de clé c
- Calcul h (c) : numéro de bloc
- Consultation de la table de hachage : récupération du bucket
- Recherche dans la page référencée par le bucket l'enregistrement ayant pour clé c.

HACHAGE

Algorithme d'insertion

- Rechercher si la nouvelle clé c n'existe pas.
- Si non : calculer le hash h(c), récupérer le bucket associé au hash.

  si la page référencé par le bucket n'est pas saturé alors

  insérer le nouvel enregistrement,

  sinon,

  allouer une nouvelle page,

  insérer le nouvel enregistrement

  chaîner la nouvelle page aux autres.

HACHAGE

Algorithme de suppression

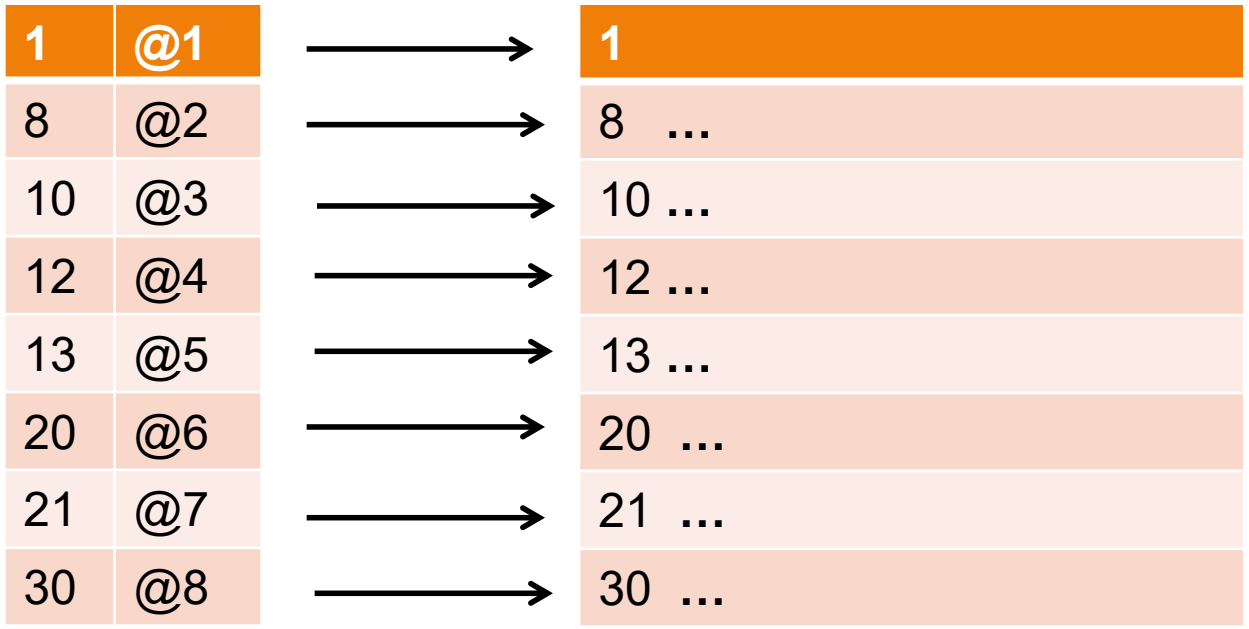
- Rechercher l'enregistrement à supprimer
- Mettre un indicateur de suppression dans l'en-tête de l'enregistrement à supprimer.
  - Si tous les enregistrements de la page sont supprimés, libérer la page qu'occupait cet enregistrement en mettant à jour le chaînage.

Fichier index

- Un fichier index contient un ensemble de couples (c, p) où c est la clé, p est l'adresse de l'enregistrement.
- Le fichier index peut être dense ou non dense:

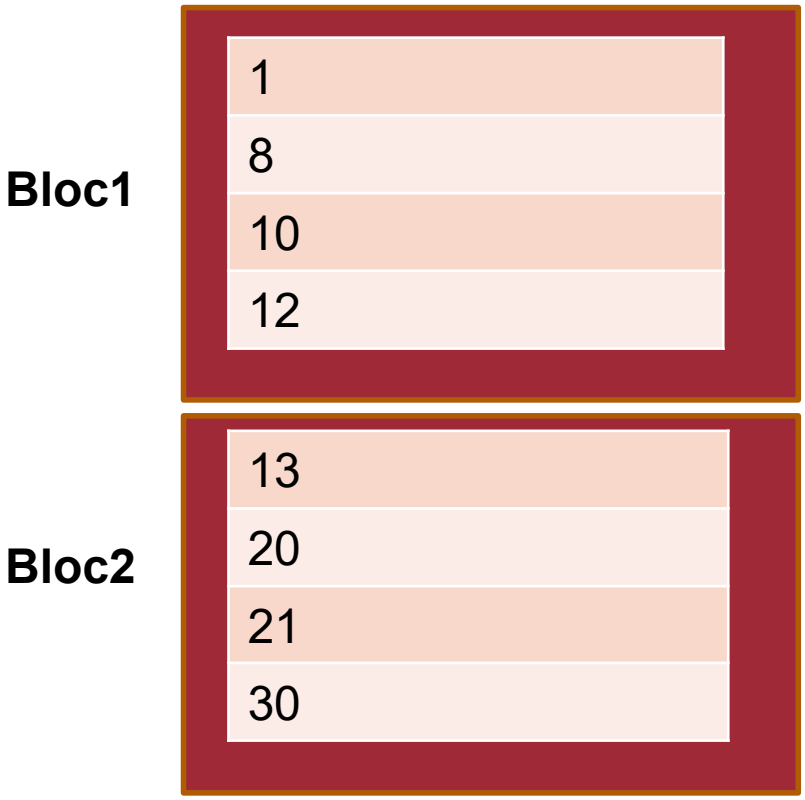
Densité = Nombre d'enregistrement indexés/nombre total d'enregistrement

- Index dense : densité = 1 : il contient toutes les clés du fichier
- Index non dense: ne contient pas toutes les clés du fichier.



Fichier index

- Index non dense : on crée des enregistrements index pour certains enregistrements du fichier : dans ce cas le fichier est trié et divisé en blocs.
- Chaque bloc lui est associée une entrée dans l'index.
- (c,p) : < plus grande clé du bloc, adresse relative du bloc>



12	@1
30	@2

INDEX NON DENSE

L'accès est direct sur l'adresse du bloc à partir de l'index puis séquentiel à l'intérieur d'un bloc

Fichier index

Algorithme de recherche

- Accès à l'index,
- Recherche dans l'index la clé d'enregistrement désirée,
- Récupération dans l'index de l'adresse relative de l'enregistrement (si index dense), ou de l'adresse relative du bloc qui le contient (si index non dense),
- Conversion de l'adresse relative en adresse réelle,
- Accès à l'enregistrement ou au bloc

Fichier index

Algorithme d'insertion :

- Accès à l'index,
- Détermination de l'emplacement de la page qui doit contenir l'enregistrement, puis détermination de la place de l'enregistrement dans la page.
- Si la place existe (page non saturée), alors insérer l'enregistrement en déplaçant les autres si nécessaire.
- Si la page est pleine, il existe différentes stratégies, entre autres, aller à la page suivante ou allouer une nouvelle page, tout en mettant à jour l'index.

Fichier index

Algorithme de suppression :

- Appliquer l'algorithme de recherche pour trouver l'enregistrement,
- Soit supprimer réellement l'enregistrement en mettant à jour l'index,
- Soit faire une suppression logique.
- Cas particuliers
  - **Si l'enregistrement à supprimer est le premier de l'index, alors une modification de l'index est nécessaire.**
  - **Lorsqu'une page devient complètement vide, il faut la rendre au système et mettre à jour l'index.**

B-arbre

- Un **B-arbre** (ou B-tree en anglais) est une structure de données arborescente auto-équilibrée
- Un B-arbre d'ordre d et de profondeur p est défini comme une arborescence ayant les propriétés suivantes :
  - Chaque nœud a au plus d fils, c'est-à-dire d pointeurs.
  - Chaque nœud excepté la racine et les feuilles a au moins [d/2] fils (arrondi supérieur de d/2).
  - La racine a au moins 2 fils.
  - Toutes les feuilles apparaissent au même niveau : (p),
  - Un nœud ayant k fils contient k-1 clés.
  - Les données (tuples) sont rangées dans les feuilles ou nœuds terminaux.

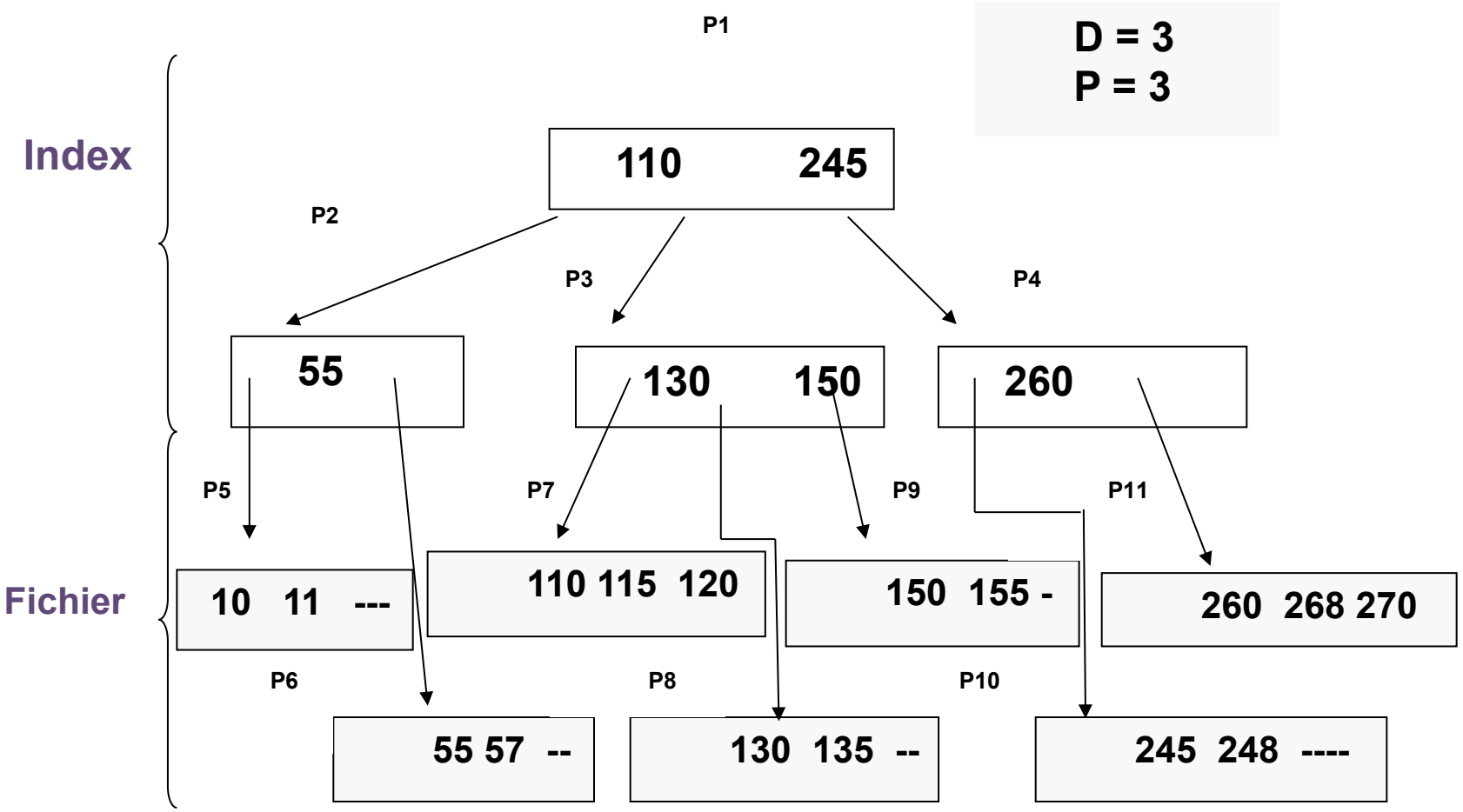
B-ARBRE : structure

- Un B-arbre a deux composantes :
  - Le **B-index** contenant les clés du fichier
  - Le **fichier** au niveau des nœuds terminaux
- Le **B-index** peut être :
  - **Un Index Principal : ou index primaire.** l'ordre physique coïncide avec l'ordre logique : il est construit au chargement initial de la relation et est défini sur un attribut clé.
  - **Un Index Secondaire :** ordre physique est différent de l'ordre logique dans la plupart des cas et peut être défini sur des attributs non clés
- L'accès est direct sur le nœud terminal à partir du B-index puis séquentiel à l'intérieur du nœud terminal

B-arbre: exemple

Par souci de clarté, seul le champ clé est représenté dans les pages du fichier (pages P5 à P11)

Hypothèse : le nœud feuille ne peut contenir plus de 3 enregistrements



B-arbre

Algorithme de recherche

Soit la recherche d'un tuple de clé K

- Commencer par la racine.

1. Parcourez les clés du nœud de gauche à droite pour trouver la première clé Ki telle que la clé recherchée K≤Ki.
2. Lire la valeur de pointeur associé.
3. Aller à la page pointée par le pointeur sélectionné.
4. Répéter les opérations 1-3 jusqu'à trouver la page feuille contenant la clé K.

B-arbre

Algorithme d'insertion

Soit à insérer un tuple de clé K

- Rechercher la page feuille du B-arbre qui doit contenir ce tuple en appliquant l'algorithme de recherche : soit Pi cette page ; deux possibilités:
- Si Pi est non saturée alors insertion dans l'ordre des clés.
- Sinon il faut allouer une nouvelle page P'. On répartit les tuples de Pi avec le nouveau tuple en deux groupes équilibrés qui seront stockés respectivement sur Pi et P'.
- Soit P0 la page qui pointait sur Pi ; deux possibilités :
- Soit P0 est non saturée, c'est-à-dire qu'elle n'a pas les d-1 clés alors insertion d'un pointeur pour P' avec la valeur minimum de clé de P'.
- Sinon, il faut allouer une nouvelle page pour l'index tout en contrôlant le pointage du niveau antécédent.
- Dans le cas où plusieurs ancêtres de P0 sont pleins alors l'insertion aura pour effet de modifier l'arbre sur plusieurs niveaux et ainsi être dans la possibilité de modifier la racine.. On parle alors d'insertion avec éclatement de nœuds et propagation de l'éclatement jusqu'à la racine.

B-arbre

Algorithme de suppression

Soit à supprimer le tuple de clé K

- Appliquer l'algorithme de recherche de la clé K : soit Pi la page qui contient K
- Si Pi a un nombre d'enregistrement >= [d/2] après suppression alors réaliser la suppression. Vérifier que la clé du tuple supprimé ne se retrouve pas comme clé dans l'index du B-arbre (vérifier tous les niveaux de l'index). Dans ce cas, il faut alors remonter dans la hiérarchie afin de remplacer cette clé par la clé du tuple suivant.
- Sinon (c'est-à-dire que Pi a moins de [d/2] fils après suppression) :
- On examine la page Pj immédiatement à gauche ou à droite de Pi et ayant le même père.
- Si Pj a plus de [d/2] enregistrements, on redistribue les enregistrements de Pi et Pj de manière équilibrée tout en conservant l'ordre (suppression avec équilibrage). On répercute la modification sur les ancêtres de Pi puisque les clés sont modifiées.
- Sinon on réalise une suppression avec fusion. On regroupe Pi et Pj en un seul bloc et on modifie les ancêtres de Pi. Cette fusion peut être récursive.

Index Bitmap

Soit un attribut A, ayant n valeurs possibles [v_1, …,v_n] (domaine)

Création d'un index bitmap sur l'attribut A:

- On crée n tableaux de bit, un pour chaque valeur v_i
- Le tableau contient un bit pour chaque tuple t
- Le bit d'un tuple t est à 1 si: t.A = v_i, à 0 sinon

```
CREATE BITMAP
INDEX Fonction_ib_idx
ON Client.Fonction
```

Select * From Employé where Fonction = Soudeur

Select count(*) From Employé where Fonction IN ('Soudeur', 'Tourneur')

Select * From Employé where Fonction IN ('Soudeur', 'Tourn

Table Employé			
ROWID(RID)	N°E	Nom	Fonction
00055 :000 :0023	1	Karim	Fraiseur
00234 :020 :8922	2	Hichem	Soudeur
19000 :328 :6200	3	Salim	Tourneur
21088 :120 :1002	4	Ali	Sableur
20088 :120 :1012	5	Lyes	Soudeur
25087 :120 :1023	6	Rabie	Tourneur

Index binaire sur l'[illegible]but Fonction				
ROWID(RID)	Soudeur	Fraiseur	Sableur	Tourneur
00055 :000 :0023	0	0	0	0
00234 :020 :8922	1	0	0	0
19000 :328 :6200	0	0	0	1
21088 :120 :1002	0	0	1	0
20088 :120 :1012	1	0	0	0
25087 :120 :1023	0	0	0	1

Soudeur		Tourneur		OR
0		0		0
1	OR	0	=	1
0		1		1
0		0		0
1		0		1
0		1		1

Questions?