Stockage des données

1. Support : mémoire

Plus une mémoire est rapide, plus elle est chère (plus elle est réduite)

- mémoire cache utilisée par le processeur pour stocker ses données et ses instructions
- mémoire vive (principale) qui stocke les données et les processus constituant l'espace de travail de la machine
- disques magnétiques, principal périphérique, qui offrent une grande capacité de stockage avec des accès en lecture et en écriture acceptables
- bandes magnétiques, supports très économiques, dont la lenteur les destine aux sauvegardes.

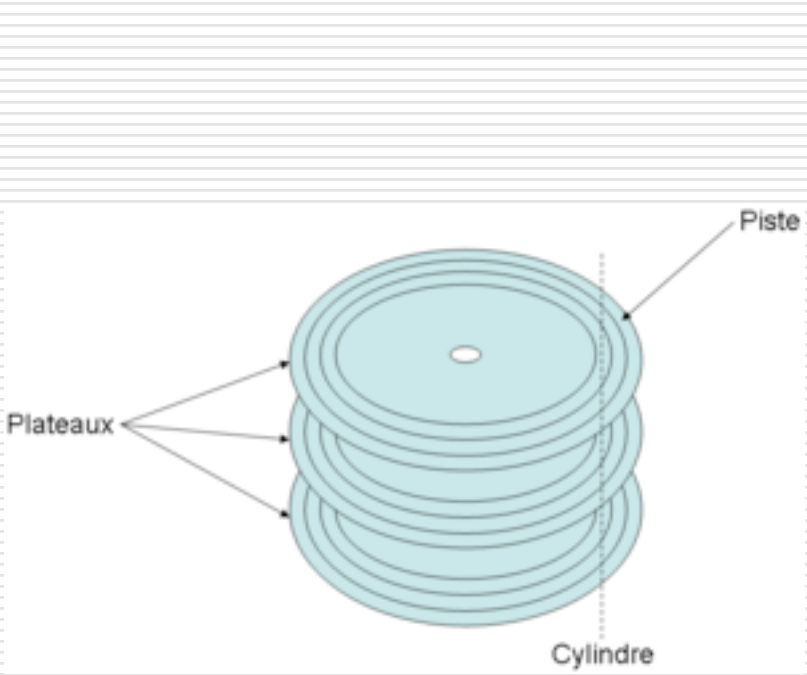
1. Support : transfert de données

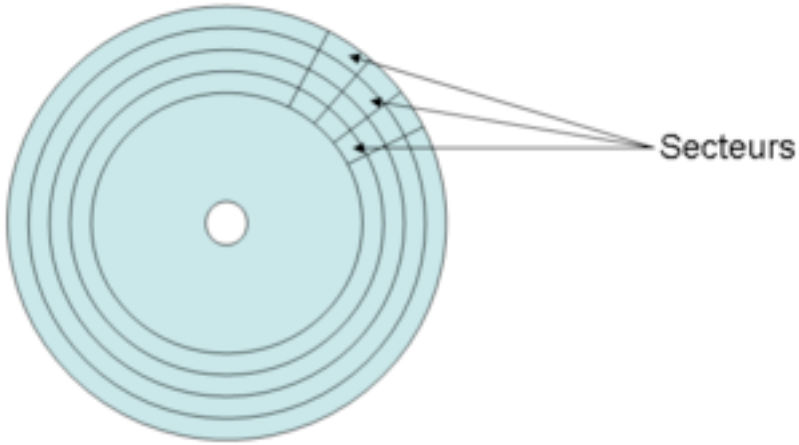
- Une base de données est toujours placée sur disque pour des raisons de taille et de persistance.
- Les données doivent impérativement être placées en mémoire vive pour être traitées.
- Un SGBD doit effectuer des transferts entre mémoire principale et disque pour satisfaire les requêtes.
- Le coût de ces transferts intervient énormément dans les performances du système.

1. Support : disque

- Un disque est une surface magnétisée (simple ou double) capable d'enregistrer des informations.
- Les disques sont divisés en secteurs : plus petite surface d'adressage. Leur taille est souvent de 512 octets.
- Une suite d'octets forme une piste sur la surface du disque.
- Le SE, au moment de l'initialisation du disque, peut fixer une unité d'entré/sortie supérieure au secteur : bloc.
- Chaque piste est divisée en blocs (pages) constituant l'unité d'échange entre le disque et la mémoire.
- Un cylindre est un ensemble de pistes de même taille.
- Toute lecture ou écriture sur les disques s'effectue par blocs.

1. Support : disque dur





2. Accès aux données

- Un disque est une mémoire à accès direct.
- Adresse donnée à chaque bloc au moment de l'initialisation du disque par le SE.
- Adresse généralement composée du n° du disque ou n° de la surface si disques à double-face, du n° de la piste et du n° du bloc sur la piste.
- Le temps de lecture varie en fonction du placement sur le disque, de l'ordre des commandes d'E/S ou de la présence des données dans une mémoire cache.
- Toutes les techniques permettant de réduire le temps d'accès sont utilisées par les SGBD.

2. Accès aux données

- Le regroupement consiste à placer dans le même bloc des données suceptibles d'être lues au même moment.
- Le placement dans des blocs contigus permet des lectures séquentielles plus performantes que les celles aléatoires.
- Les SGBD essaient d'optimiser la proximité des données au moment de leur placement sur le disque.
- Une table est stockée sur une même piste ou sur les pistes d'un même cylindre pour un parcours séquentiel efficace.
- Le SGBD doit avoir, à la création de la base, un espace important sur le disque dont il sera le seul à gérer l'organisation.
- Une mémoire cache (ou buffers ou tampons) est un ensemble de blocs en mémoire principale qui sont des copies des blocs sur le disque.
- L'administration d'une base de données consiste en partie à spécifier la mémoire qui peut être attribuée en permanence au SGBD. Plus cette mémoire est grande et plus la performance sera importante.

3. Fichiers

- Un fichiers est composé d'enregistrements.
- Quelques informations sont stockées dans un en-tête : taille de l'enregistrement (si variable), pointeur vers le schéma de la table (pour savoir son type), date de la dernière mise à jour, …
- Les SE organisent les fichiers dans une arborescence de répertoires.
- Il faut distinguer l'emplacement physique du fichier sur le disque et son emplacement logique dans l'arbre des répertoires.

3. Fichiers : adressage

- Le stockage des enregistrements dans un fichier tient compte du découpage en blocs de ce fichier.
- On peut placer plusieurs enregistrements dans un bloc et mais un enregistrement ne chevauche pas 2 blocs.
- On peut localiser physiquement un enregistrement par son fichier, puis par le bloc, puis par la position dans le bloc.
- Pour permettre le déplacement des enregistrements on combine une adresse logique qui identifie un enregistrement indépendamment de sa localisation.
- Pour localiser un enregistrement, on donne l'adresse physique de son bloc, puis dans le bloc, on gère une table de localisation au sein du bloc ou éventuellement dans un autre bloc.

3. Fichiers : indexation

- La création d'un index permet de créer des chemins d'accès aux enregistrements plus directs.
- Un index permet de satisfaire certaines requêtes (mais pas toutes) portant sur un ou plusieurs attributs (mais pas tous).
- L'index existe indépendamment de l'organisation du fichier, ce qui permet d'en créer plusieurs pour optimiser plusieurs types de requêtes.
- La création d'un nombre important d'index est pénalisante pour le SGBD qui doit gérer, pour chaque mise à jour sur une table, la récupération de cette mise à jour sur tous les index de la table.
- Un index est une structure permettant d'optimiser les recherches par critères de recherche (souvent sur clé primaire)
  - le parcours de l'index fournit l'adresse de l'enregistrement,
  - par accès direct au fichier on obtient l'enregistrement.

4. Indexation : index non-dense

- Dans le cas d'un fichier trié sur la clé primaire, il est possible d'effectuer une recherche dichotomique, mais le fichier est souvent fragmenté car très gros.
- Dans ce cas, l'index est lui-même un fichier contenant des enregistrements [valeur, adresse].
- Toutes les valeurs de clé existant dans le fichier de données ne sont pas représentées dans l'index (index non-dense).
- L'index n'a que les valeurs de clé des premiers enregistrements de chaque bloc. La recherche dichotomique est effectuée sur le fichier d'index. Une seule lecture suffit pour trouver l'enregistrement.
- Un index non-dense est très efficace pour les opérations de recherche mais le problème est de maintenir l'ordre du fichier pour les insertions et les destructions. Ce type d'index est peu utilisé par les SGBD.

5. Indexation : index dense

- Si on veut indexer un fichier qui n'est pas trié sur la clé de recherche, il faut baser l'index sur toutes les valeurs de clé existant dans le fichier et les associer à l'adresse d'un enregistrement (et pas à l'adresse d'un bloc).
- Un index dense peut coexister avec un index non-dense. On peut trier un fichier sur la clé primaire et créer un index non-dense, puis ajouter des index denses pour les attributs qui servent fréquemment de critère de recherche.

5. Indexation : index multi-niveaux

- Il peut arriver que la taille du fichier d'index soit si grande que les recherches dans l'index soient pénalisées.
- La solution est d'indexer le fichier d'index lui-même.
- Index multi-niveaux permet de passer, dès le second niveau, d'une structure dense à une structure non-dense.
- Très efficaces en recherche même pour des données de très grande taille. La difficulté est de maintenir des fichiers triés sans dégradation des performances.

5. Indexation : B-arbre

Arbre binaire généralisé équilibré

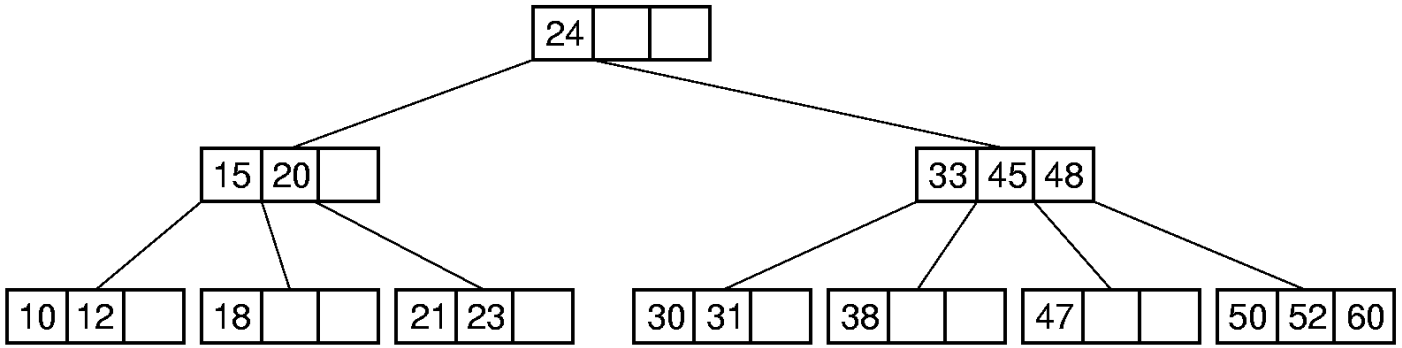
- Blocs chaînés entre eux pour créer une structure arborescente
- Chaque nœud est contenu dans une page de disque (une seule opération)
- Chaque nœud du B-arbre contient
  - un tableau de clés trié par ordre croissant,
  - un tableau d'enregistrements associés à ces clés,
  - n : le nombre de clés stockées dans le nœud,
  - un booléen valant vrai si le nœud est une feuille, faux sinon,
  - si le nœud est interne, un tableau de pointeurs vers un sous-arbre dont toutes les clés k sont tq $\forall i \in [2, n]$, clé[i-1]≤k≤clé[i] ; k≤clé[1] et clé[n]≤k.
- Tout chemin de la racine à une feuille a une longueur h (hauteur).

  Le degré minimum du B-arbre t (t≥2) fixe le nombre minimum et maximum de clés par nœud : tout nœud non racine contient un nombre de clés n tel que t-1≤n≤2t-1.

  Tout nœud interne possède entre t et 2t fils (2, 3 ou 4 fils pour le plus simple).

5. Indexation : B-arbre

- Exemple



5. Indexation : B+-arbre

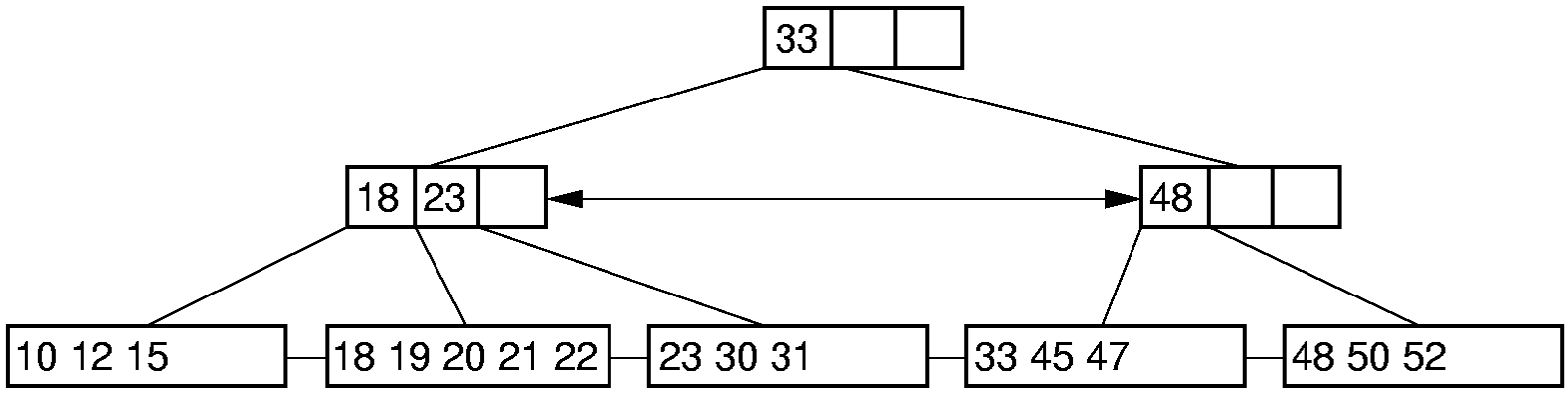
- Un B+-arbre est un B-arbre où les feuilles contiennent toutes les clés (dernier niveau)
- Les feuilles sont liées entre elles dans l'ordre lexicographique des valeurs

  → permet de répondre aux recherches par intervalle
- Les recherches, insertions et suppressions dans un B+-arbre s'effectuent par le parcours d'un chemin de l'arbre

  → nombre d'accès disque = O(h) = O(log(n))
- Les B+-arbres sont grandement utilisés pour les bases de données

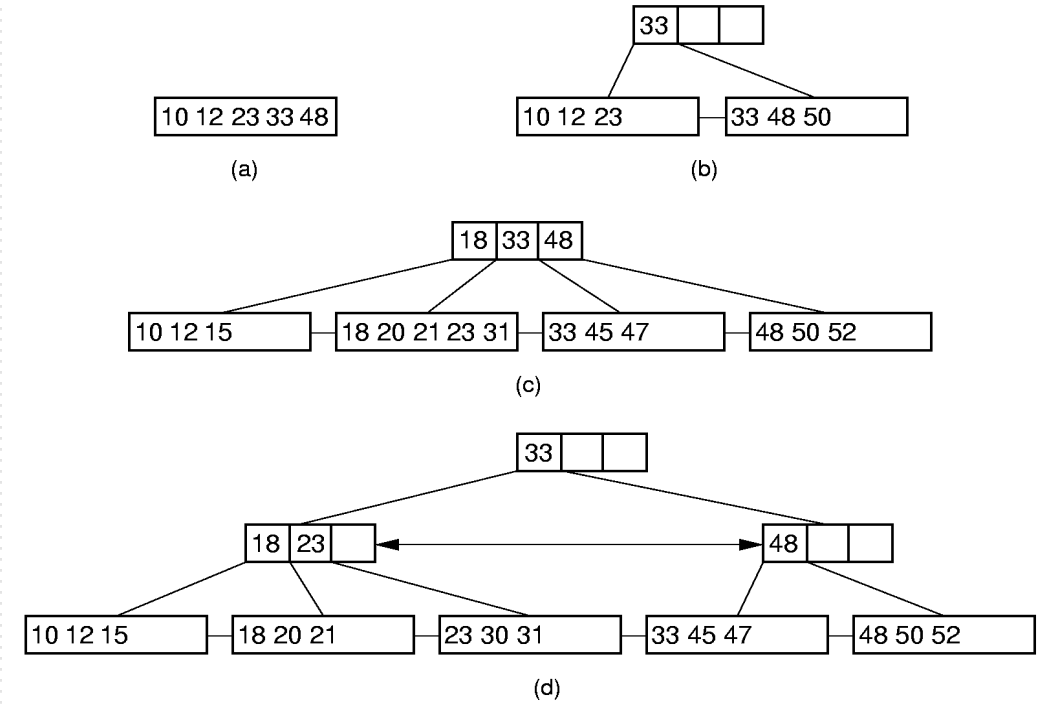
5. Indexation : B+-arbre

- Exemple



5. Indexation : B+-arbre

Insertions



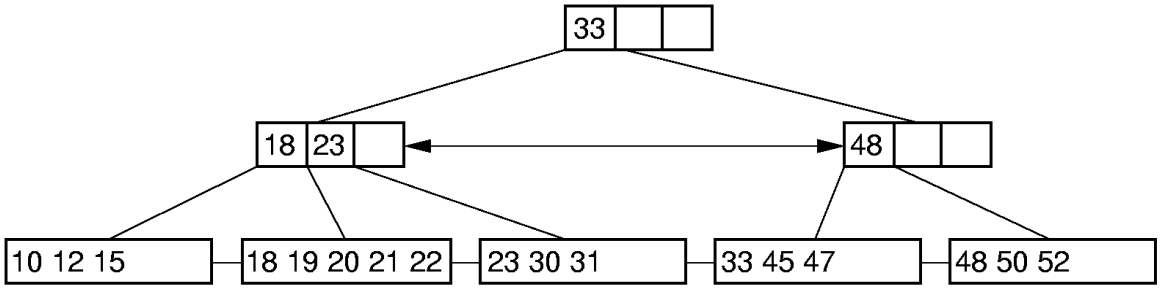
Insertion de 50

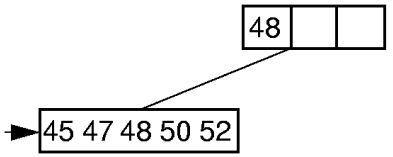
Plusieurs insertions

Insertion de 30

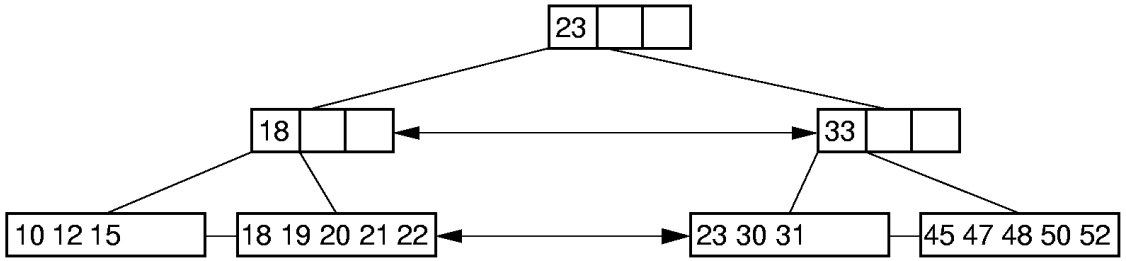
5. Indexation : B+-arbre

- Suppression de 33





(a)



(b)

5. Indexation : hachage

- Organiser des données d'après une clé et utiliser une fonction qui, pour chaque valeur de clé c, donne l'adresse f(c) d'un espace de stockage.
- Pour obtenir une distribution uniforme avec la fonction d'adressage, on utilise les 4 ou 8 premiers caractères de la clé comme des entiers qu'on somme.
- Indirection entre l'identification « logique » du bloc et son emplacement physique.
- Avantages
  - la structure n'occupe aucun espace disque (contrairement au B-arbre), le répertoire tient en mémoire principale,
  - permet d'effectuer les recherches par clé par accès direct (calculé) au bloc susceptible de contenir les enregistrements.
- Inconvénient : ne permet pas d'optimiser les recherches par intervalle puisque l'organisation des enregistrements ne s'appuie pas sur l'ordre de clés.

5. Indexation : index bitmap

- Un index bitmap considère toutes les valeurs possibles pour un attribut, qu'elle soit présente ou non dans la table.
- Pour chacune de ces valeurs, on stocke un tableau de bits (bitmap) avec autant de bits qu'il y a de lignes dans la table
  - Si le bit est à 1, l'attribut A a pour valeur v dans la ligne l
  - Sinon le bit est à 0.
- Pour rechercher une valeur v, il faut prendre le bitmap associé à v, chercher tous les bits à 1 et accéder aux enregistrements.
- Un index bitmap est très efficace si le nombre de valeurs possibles pour un attribut est faible.