Université Constantine 2
جامعة قسنطينة 2

Développement Avancé d'Applications Web

– Chapitre 2 –

Gestion de la Mémoire & l'Indexation dans les BDs

Dr Bouanaka Chafia

NTIC

chafia.bouanaka@univ-constantine2.dz

Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
Nouvelles technologies	TLSI	Licence 3	Génie Logiciel (GL)

Résumé

Objectifs du cours

- Comprendre les principes et les techniques de stockage des données
- Comprendre la notion d'indexation dans les BDs.
- Maitriser les techniques d'indexation dans les BDs

Plan du Cours

- Organisation physique des données
  - Stockage des données sur disque
  - Traitement des données
- Indexation dans les BDs
  - Notion d'index
  - Typologie des index
  - Techniques d'indexation

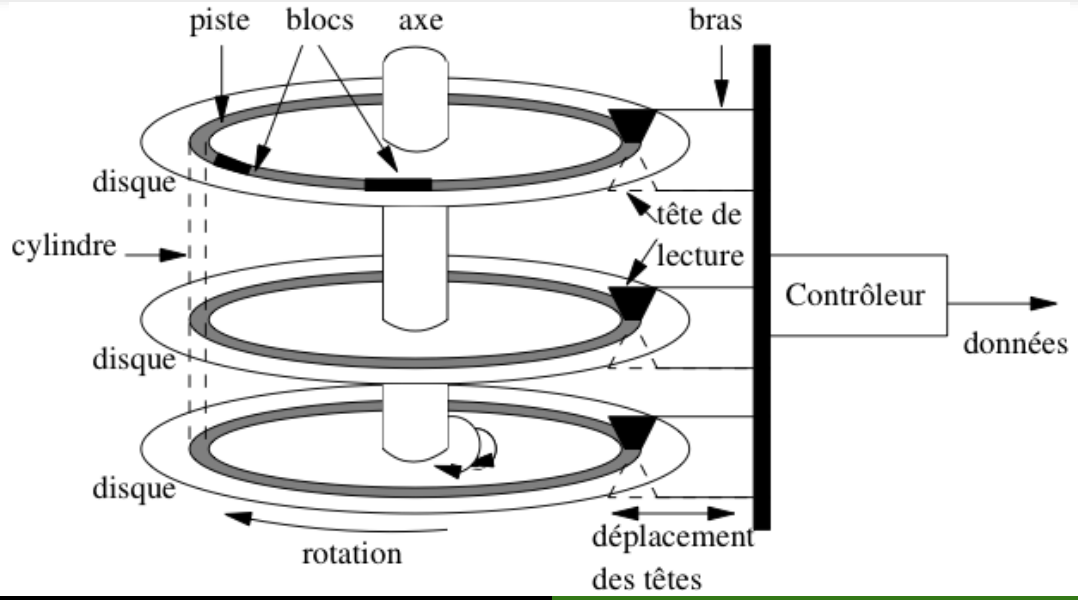
Section 1 :
Organisation Physique des données

Organisation physique des données

Définitions

Stockage d'une BD

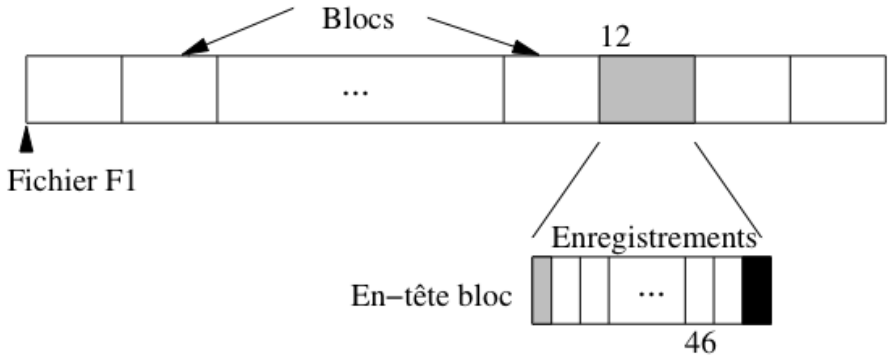
- La notion **d'organisation des données** concerne la façon dont les enregistrements sont rangés dans un fichier.
- Une **base de données** est organisée en un ensemble de données stockées sur un support persistant (Disque dur : voir figure ci-dessous).
- Le stockage d'une BD consiste à organiser les données sur un disque au moyen de fichiers. Chaque **fichier** correspond à **une table** dans la BD.



Organisation physique des données

Enregistrement

- Pour le système d'exploitation, **un fichier** est une suite d'octets répartis sur un ou plusieurs **blocs**.
- Les fichiers gérés par un SGBD sont un peu plus structurés.
- Ils sont constitués d'**enregistrements** qui représentent physiquement les entités du SGBD.
- Selon le modèle logique du SGBD, ces entités peuvent être :
  - des tuples (pour un SGBDR) dans une relation,
  - ou des objets (pour une BD orientée objet).
- Un **tuple** dans une table relationnelle est constitué d'une liste d'attributs, chacun ayant un type.
- À ce **tuple** correspond un **enregistrement**, constitué de champs.
- Chaque type d'attribut détermine la taille du champ nécessaire pour stocker une instance du type.



Organisation physique des données

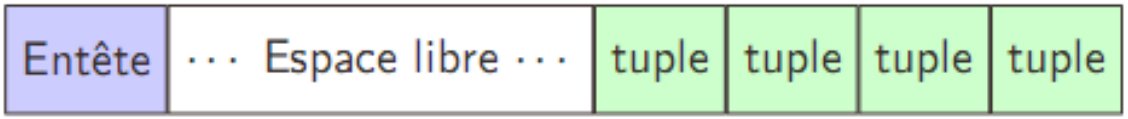
Blocs

- Le stockage des enregistrements dans un fichier doit tenir compte du découpage en blocs de ce fichier.
- En général il est possible de placer plusieurs enregistrements dans un bloc.
- Il est nécessaire d'éviter qu'un enregistrement soit sur deux blocs.
- Le nombre maximal d'enregistrements de taille **E** pour un bloc de taille **B** est donné par **B/E**

Organisation physique des données

Blocs

- Chaque accès, écriture dans la base de données se fait à partir du bloc (page disque).
- Il est structuré pour pouvoir rapidement récupérer les données à l'intérieur :



Les tuples peuvent être :

- de taille fixe
- de taille variable

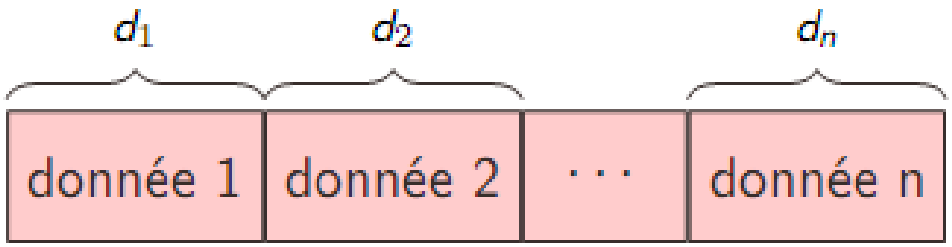
Organisation physique des données

Tuples avec données à taille fixe

- Si un tuple est constitué de données de tailles fixes de types suivants :

  (t, . . . tn)

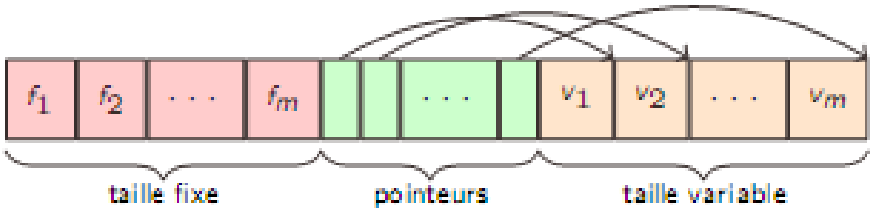
- de tailles respectives d1, . . . , dn, il suffit de mettre les champs les uns derrière les autres :



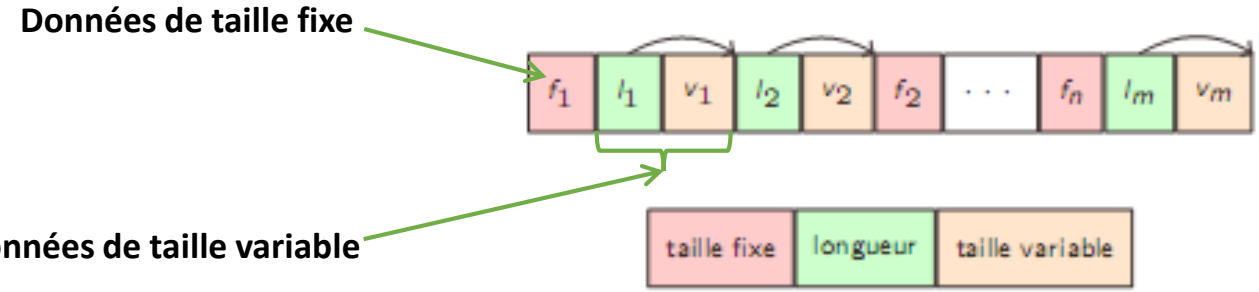
Organisation physique des données

Tuple avec données à taille variable

- Si un tuple contient des données de tailles variables, il y a plusieurs représentations possibles :
  - placer toutes les données de tailles fixes d'abord, puis les données de tailles variables en les faisant précéder des pointeurs pour trouver la donnée suivante.



  - placer sa longueur devant chaque champs de taille variable :

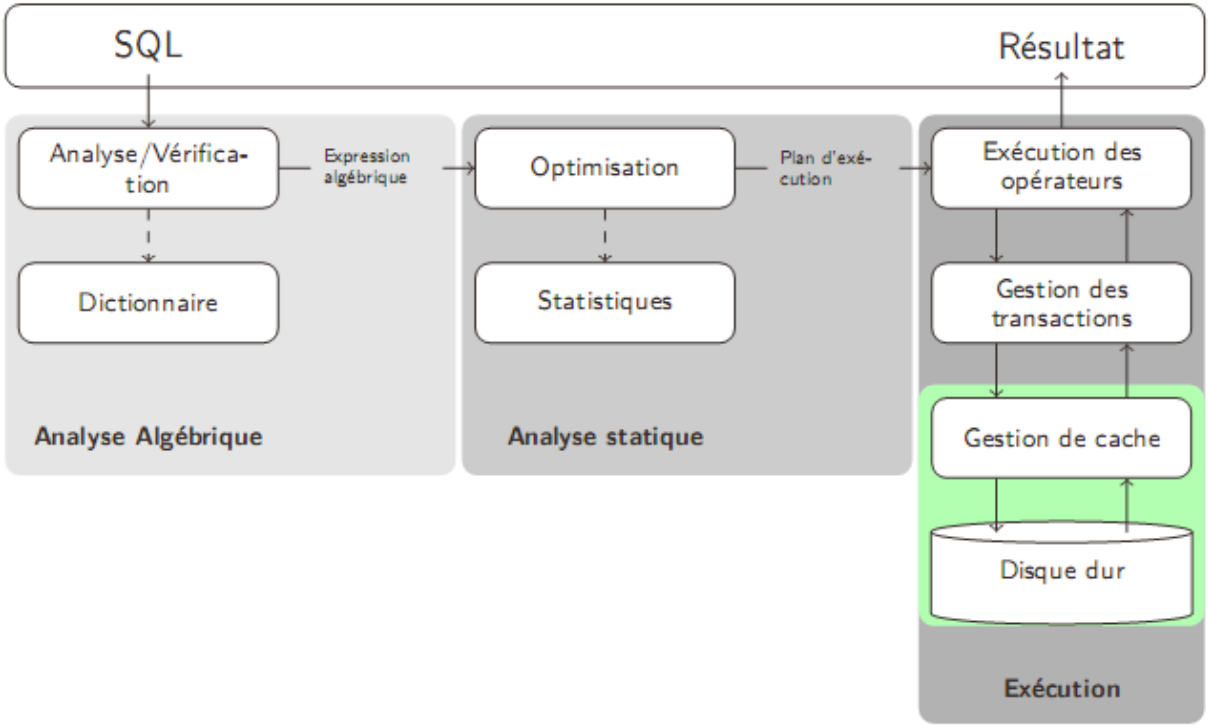


Remarque : Le SGBD associe à chaque tuple une adresse physique pour pouvoir le retrouver

Organisation physique des données

Traitement des données

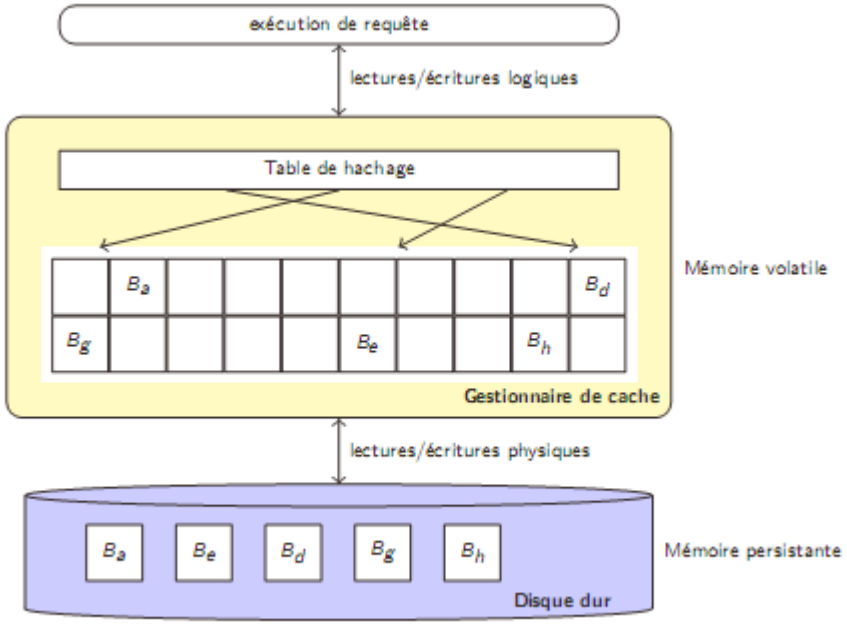
- Le traitement des données nécessite de les transférer de **la mémoire secondaire** (mémoire lente) ou le disque vers une mémoire plus rapide (**mémoire cache ou tampon**).
- Le temps de transfert des données depuis la mémoire secondaire (disque dur) est le plus important, c'est lui qui limite la chaîne de traitement.
- Généralement, on néglige le temps de calcul par rapport à ce temps de transfert.



Organisation physique des données

Le gestionnaire de cache

- Les SGBD utilisent une **mémoire tampon** afin de conserver une image des blocs sur le disque et accélérer les accès aux tuples.
- Cette mémoire tampon peut être paramétrée, sa taille peut être un facteur essentiel de l'efficacité d'un SGBD.



Section 2 :
Indexation dans les BDs

Déroulement d'une lecture logique

- Lors de la demande d'une donnée pour la traiter par un SGBD, il y'a deux possibilités :
  - le bloc qui la contient est chargé dans le cache, la donnée est directement retournée,
  - sinon, il faut lire le bloc sur le disque et le charger dans le cache
- Il y a cependant deux situations possibles :
  - le cache contient encore de l'espace libre, et il n'y a pas de problème,
  - le cache est saturé et il faut libérer un espace à l'intérieur

- une bonne gestion du cache doit maximiser le **hit ratio** :

$$\boldsymbol{hit\ ratio} = \frac{nb\ Lectures\ Logiques\ - nb\ Lectures\ Physiques}{nb\ Lectures\ Logiques}$$

Remarque : Pour maximiser le hit ratio, il faut réduire le nombre de lectures physiques, c-à-d, réduire le nombre d'accès disque.

Indexation dans les BDs

Déroulement d'une lecture logique

- Le parcours complet d'une table peut prendre un temps très important,
- pour accélérer l'accès aux données, il convient de maintenir des structures de données qui limitent le nombre de lectures sur le disque,
- en particulier, il s'agit de trouver rapidement les tuples qui ont un champs ayant une certaine valeur,
- ces structures de données portent le nom d'**index**.

Pour rechercher une information dans un fichier, il peut être parcouru :

- **Séquentiellement**: Aucun index n'est défini sur le fichier
- **Par index** : afin d'optimiser la recherche et retrouver l'information rapidement, un **fichier index** est crée.

Indexation dans les BDs

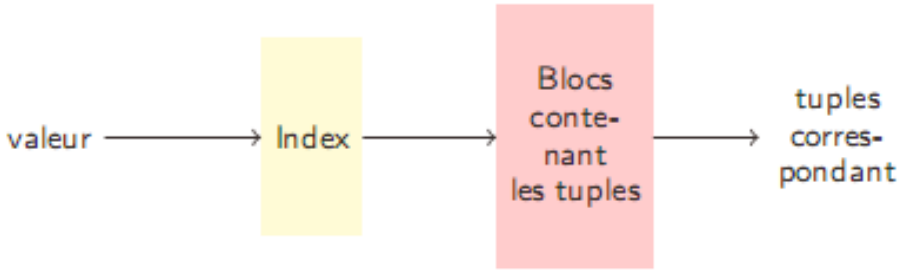
Notion d'index

Les définitions suivantes concernent l'utilisation d'un index :

- **Clé d'indexation** = une liste d'un ou plusieurs attributs.
- **Une adresse** est une adresse de bloc ou une adresse d'enregistrement.
- **Entrée d'index** : enregistrements de la forme **[valeur, addr]**, valeur est une valeur de clé.
- L'index est trié sur **valeur**

La recherche indexée est réalisée en deux étapes :

- Recherche dans le fichier index : retrouver l'adresse du bloc ou l'enregistrement recherché
- Accès au fichier de données en utilisant l'adresse retrouvée dans l'étape précédente



Indexation dans les BDs

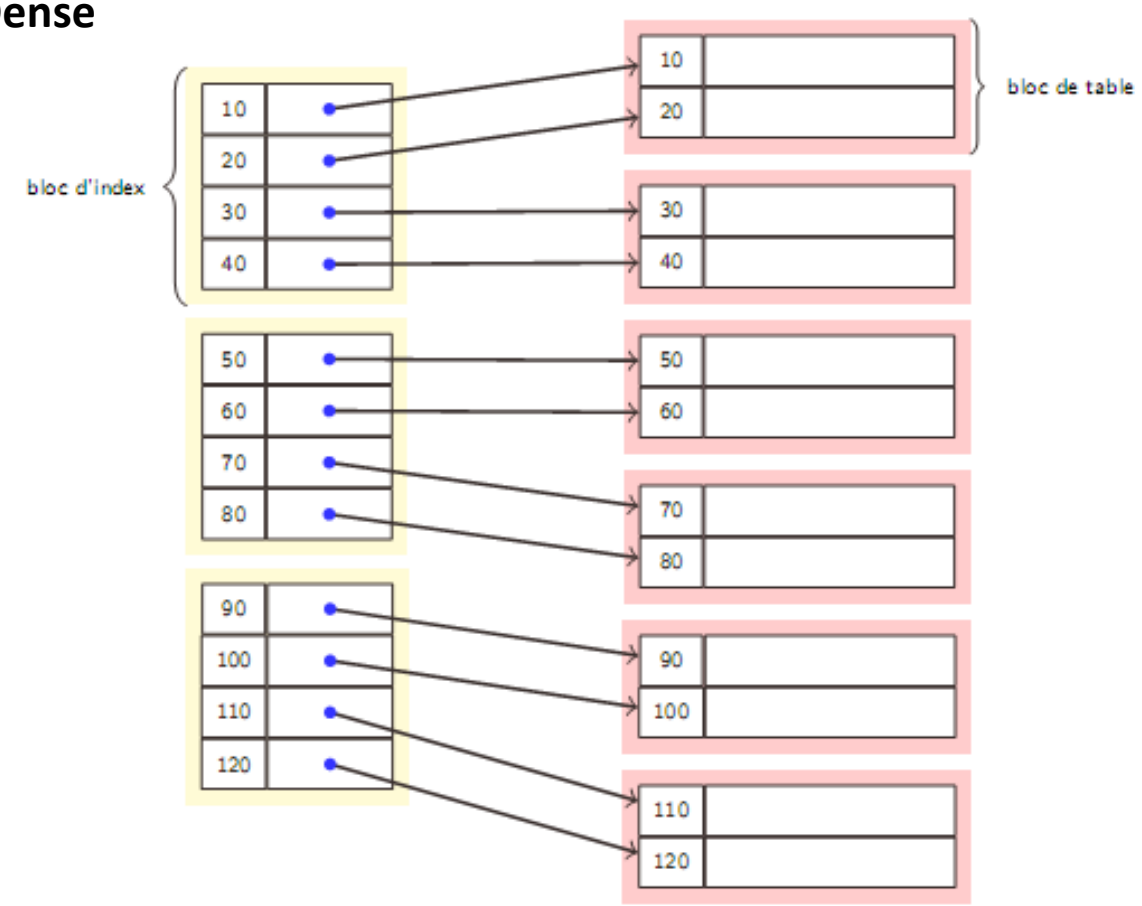
Typologie des index

- **Index primaire :** un index primaire requiert que la relation soit ordonnée, les index primaires sont typiquement utilisés sur les clés primaires (mais pas nécessairement) ;
- **Index secondaire :** l'organisation des données sur disque est indépendante de l'index.
- **Index dense :** il y a **une entrée** par **valeur possible**,
- **Index non-dense :** il y a **une entrée** par **bloc**.

	Primaire	Secondaire
Dense	Fichier ordonné, chaque tuple est indexé	Organisation des données quelconque (ordonnées ou non), chaque tuple est indexé
Non-Dense	Fichier ordonné, chaque bloc est indexé	Ce type d'index ne fait pas de sens. Il n'existe pas.

Typologie des index

Exemple 1 : Index Dense

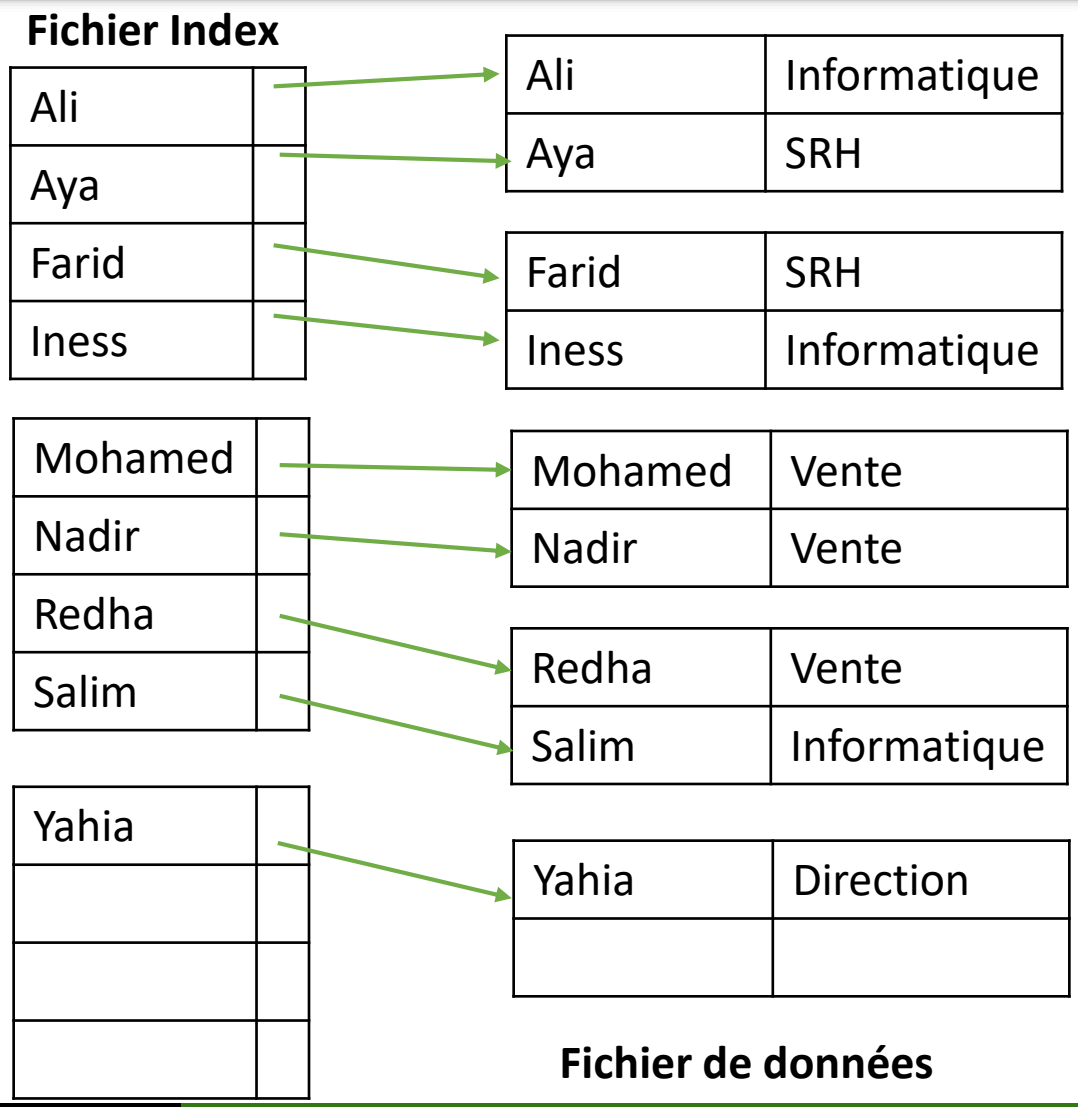


Indexation dans les BDs

Typologie des index

Exemple 2: Index Dense

Nom	Département
Ali	Informatique
Mohamed	Vente
Nadir	Vente
Salim	Informatique
Yahia	Direction
Aya	SRH
Iness	Informatique
Farid	SRH
Redha	Vente

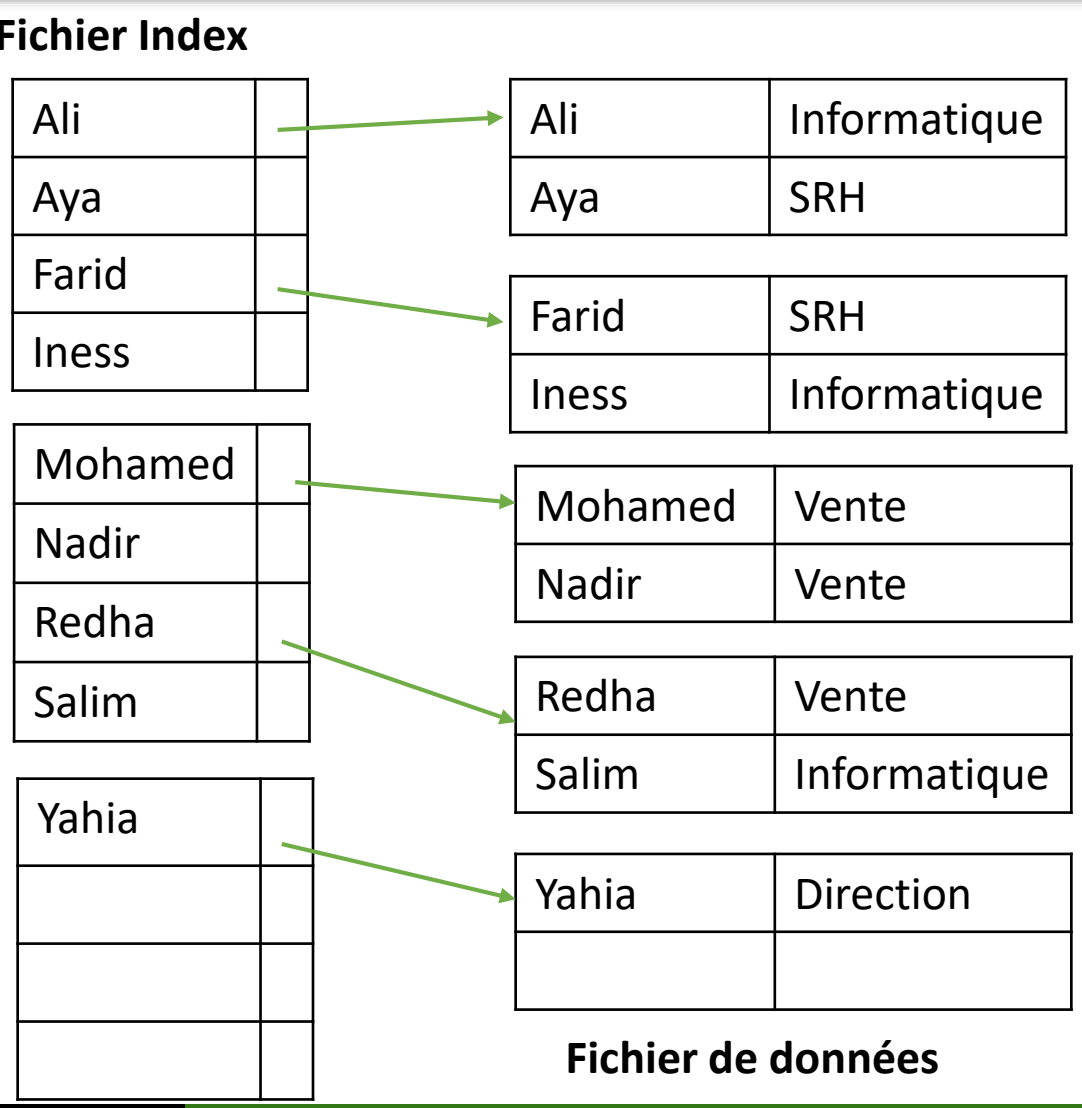


Indexation dans les BDs

Typologie des index

Exemple 3: Index non Dense

Nom	Département
Ali	Informatique
Mohamed	Vente
Nadir	Vente
Salim	Informatique
Yahia	Direction
Aya	SRH
Iness	Informatique
Farid	SRH
Redha	Vente



Indexation dans les BDs

Index Secondaire

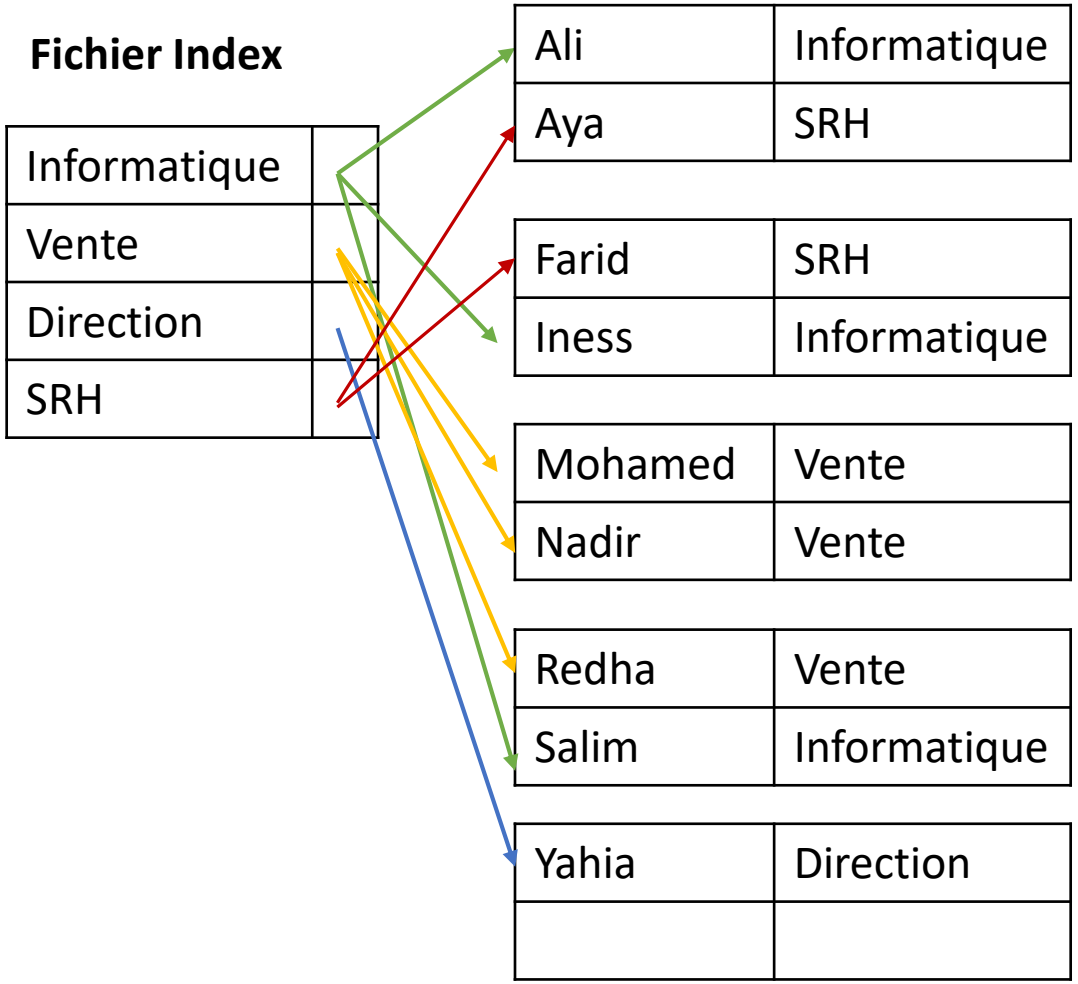
Les index secondaires servent essentiellement à retrouver des tuples en connaissant la valeur d'un champs qui n'est pas une clé primaire
Contrairement à la clé primaire, il y'a plus d'un tuple associé à une valeur d'un index secondaire

Indexation dans les BDs

Typologie des index

Exemple: Index secondaire

Nom	Département
Ali	Informatique
Mohamed	Vente
Nadir	Vente
Salim	Informatique
Yahia	Direction
Aya	SRH
Iness	Informatique
Farid	SRH
Redha	Vente



Recherche dans un fichier indexé

Un index primaire suppose que la relation est stockée ordonnée, en général selon sa clé primaire.
Le fichier est trié par rapport à sa clé.

- Les enregistrements sont groupés par blocs.
- Accès direct au bloc contenant un enregistrement.
- parcours séquentiel aux enregistrement du bloc

Recherche : la recherche d'une valeur se fait en deux étapes :
Soit à chercher une clé C:

1. Chercher dans le fichier index un rang *i* tel que:

$$Clé_{i-1} < C < Clé_i$$

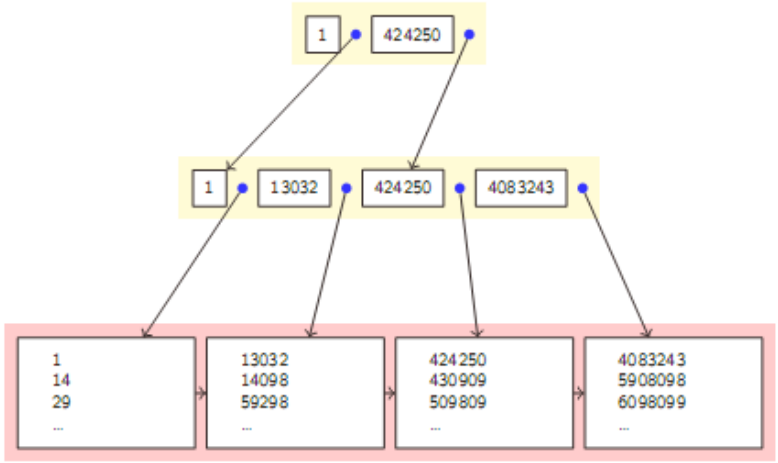
On a alors, pid_i qui est le n° de bloc où C devrait se trouver

2. Chercher dans le fichier indexé dans le bloc pid_i l'information relative à C.
Si C n'existe pas, C n'appartient pas au fichier

Recherche dans un fichier indexé

Recherche dans le fichier index

- Si l'index est ordonné dans le fichier index, on peut utiliser une recherche dichotomique
- Sinon, on peut effectuer un Full Scan ou un parcours séquentiel du fichier index
- ou encore créer un nouvel index non dense, pour indexer le premier index et donc obtenir **un index à plusieurs niveaux** (voir figure ci-dessous)

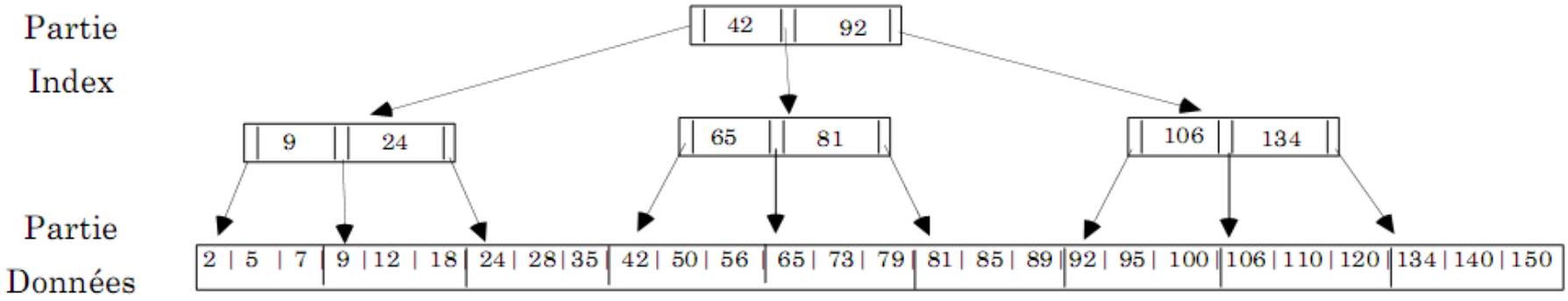


D'autres organisations sont possibles pour améliorer la recherche dans un fichier index

Arbre de recherche B+

Les B-arbres sont des structures d'index qui généralisent plusieurs idées :

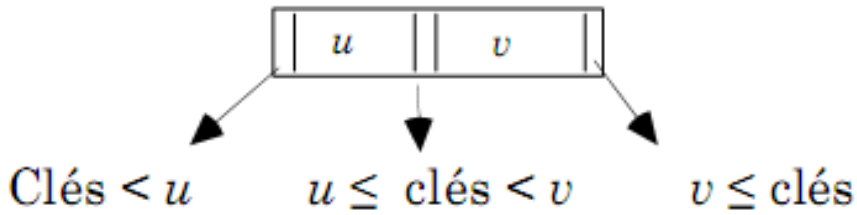
- celle des index à plusieurs niveaux,
- Ce sont des arbres équilibrés afin de garantir une recherche qui requiert un temps logarithmique dans le nombre de valeurs indexées.
- Un arbre équilibré est un arbre dont les hauteurs des sous-arbres gauche et droit différent au plus de la valeur entière un (1)



Arbre de recherche B+

Un arbre binaire de recherche satisfait aux critères suivants :

- L'ensemble des étiquettes est totalement ordonné.
- Une étiquette est appelée clé.
- Les clés de tous les nœuds du sous-arbre gauche d'un nœud ***u***, sont inférieures à la clé de ***u***.
- Les clés de tous les nœuds du sous-arbre droit d'un nœud ***u***, sont supérieures ou égales à la clé de ***u***.



Arbre de recherche B+

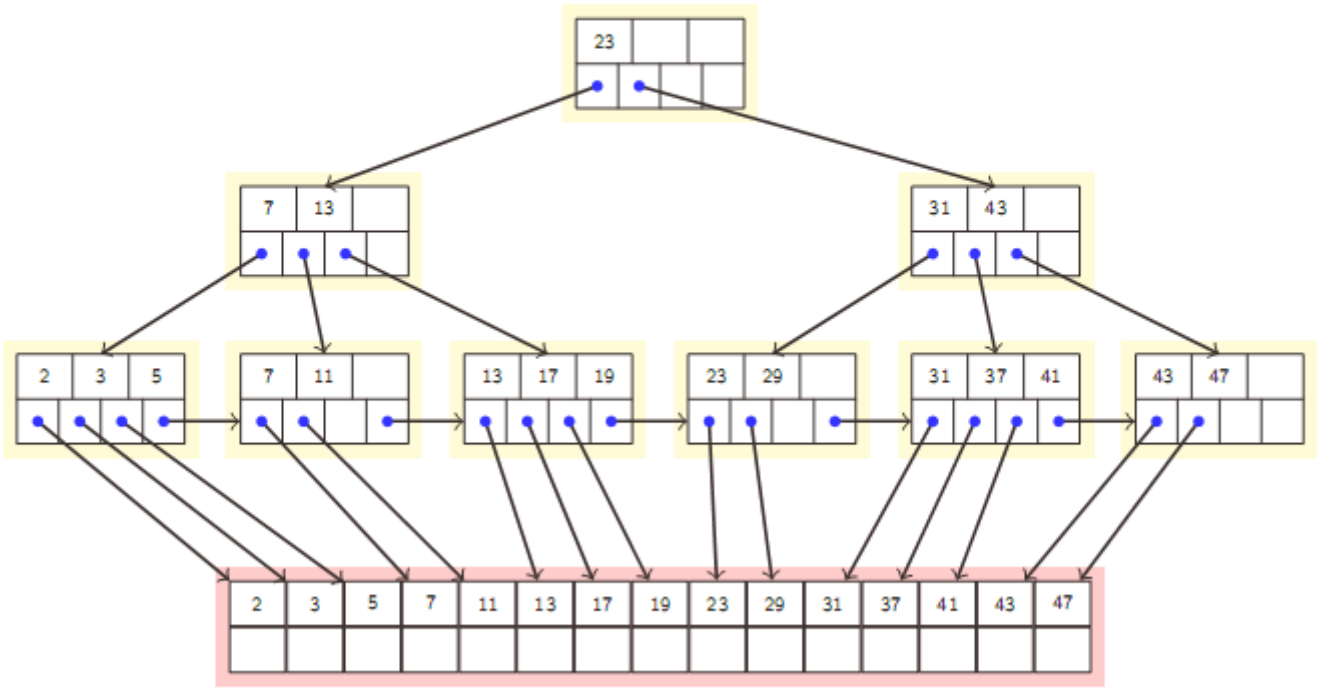
Définition Arbre Binaire

Un arbre-B d'ordre ***m*** est un arbre tel que :

- Chaque nœud contient k clés triées avec : $m \leq k \leq 2m$ (**nœud non racine**) et $1 \leq k \leq 2m$ (**nœud racine**).
- Chaque chemin de la racine à une feuille est de même longueur
- Un nœud est :
  - Soit terminal (feuille)
  - Soit possède $(k+1)$ fils tels que les clés du $i^{ème}$ fils ont des valeurs comprises entre les valeurs du $(i-1)^{ème}$ et $i^{ème}$ clés du père

Arbre B+

Exemple : $n = 3$



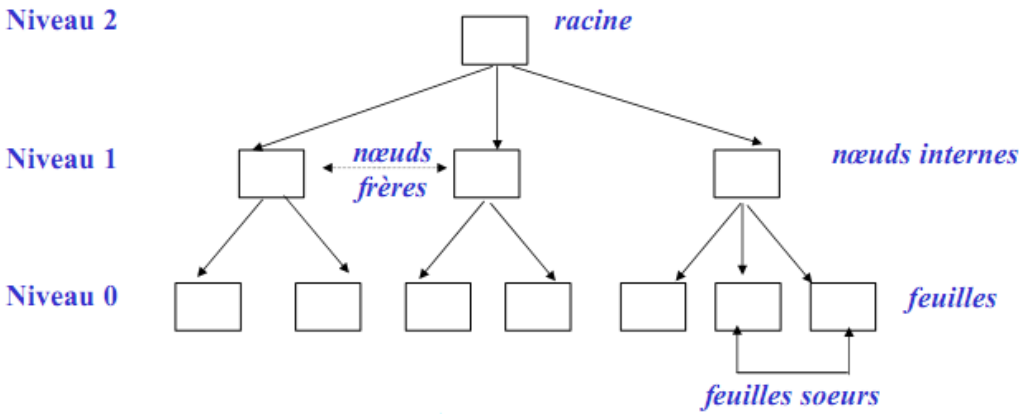
Arbre B+

Dans un arbre B+ d'ordre m :

- Tout nœud d'index a au maximum *m* nœuds fils
  - un nœud possède au minimum [*m/2*] fils
  - la racine possède au minimum *2* fils
  - tout nœud d'index contient *k* fils et (*k-1*) clés

L'arbre est équilibré (**balanced tree** ou **B+**), si :

- tous les nœuds feuilles sont au même niveau
- la hiérarchie de l'arbre grossit par la racine :
  - tous les chemins de la racine aux nœuds feuilles ont la même longueur



Arbre de recherche B+ : Recherche

Recherche : Soit C la clé à chercher. N représente un noeud:

N = racine ;

Répéter

si ***C < N.u*** alors

N = N.FilsG;

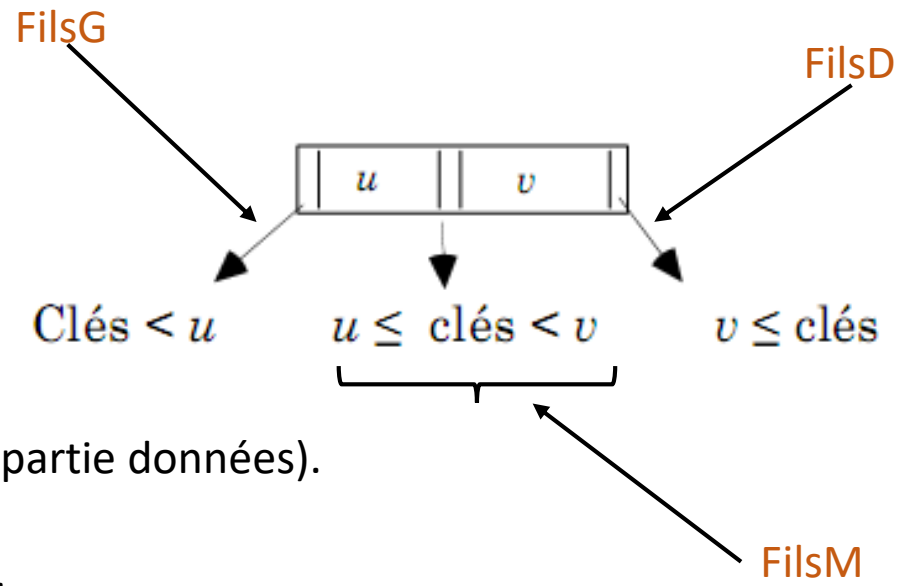
si ***N.u ≤ C < N.v*** alors

N = N.FilsM;

si ***N.v ≤ C*** alors

N = N.FilsD;

Jusqu'à N est un nœud feuille (dans partie données).



Chercher C dans le bloc ainsi trouvé.

Si n'existe pas alors C ∉ au fichier

Arbre de recherche B+ : Recherche

Recherche : Soit C la clé à chercher. N représente un noeud:

N = racine ;

Répéter

si ***C < N.u*** alors

N = N.FilsG;

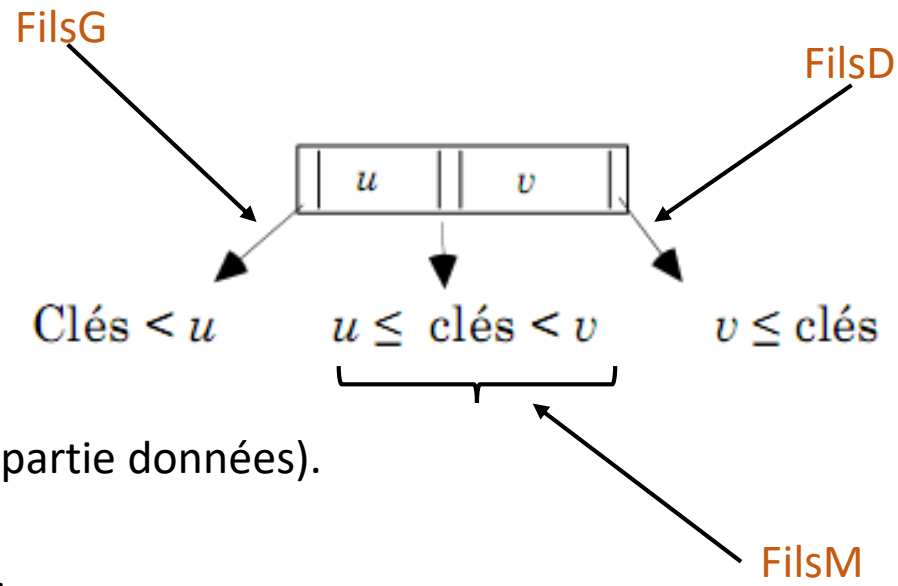
si ***N.u ≤ C < N.v*** alors

N = N.FilsM;

si ***N.v ≤ C*** alors

N = N.FilsD;

Jusqu'à N est un nœud feuille (dans partie données).



Chercher C dans le bloc ainsi trouvé.

Si n'existe pas alors C ∉ au fichier

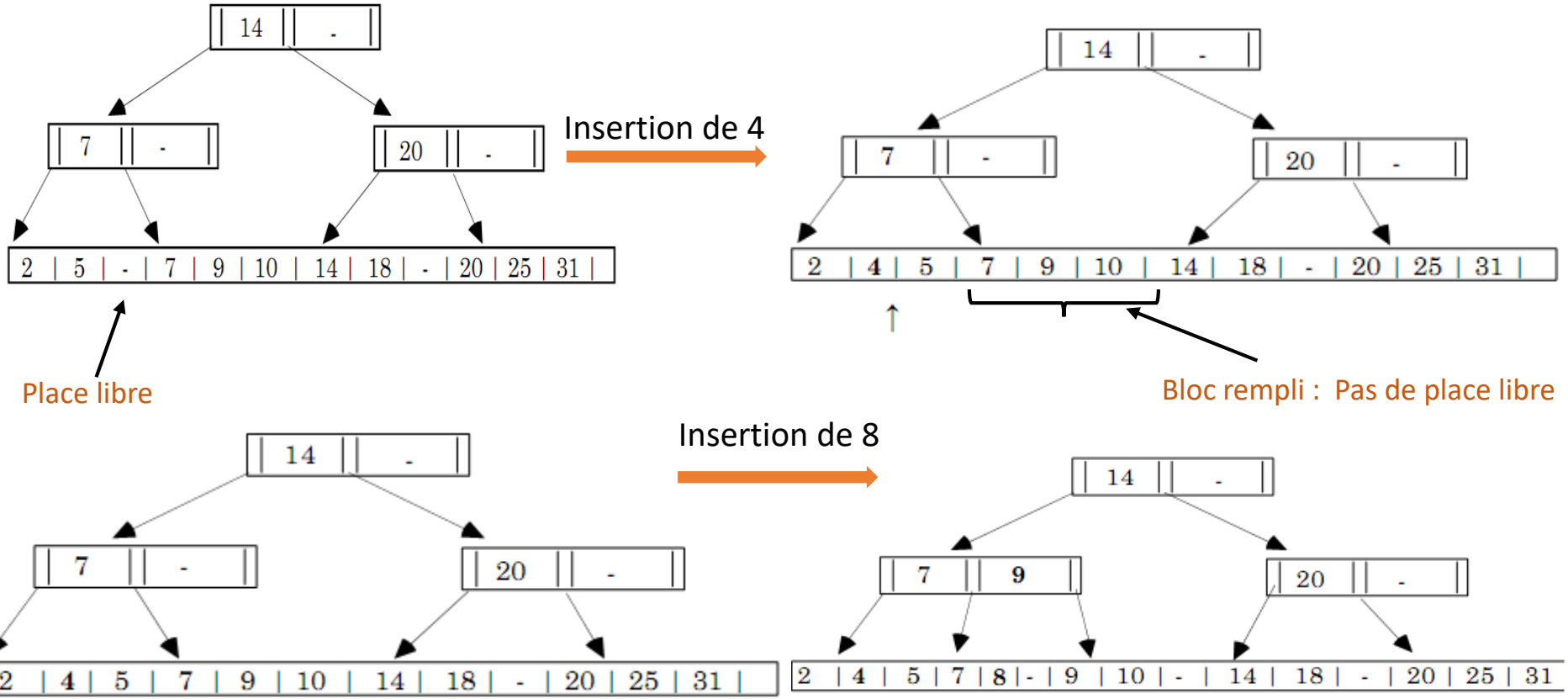
Arbre de recherche B+ : **Insertion**

Insertion :

- on essaie de placer la paire (valeur, pointeur) dans la feuille appropriée,
- si il n'y a pas de place, on découpe la feuille en deux :
  - on a $n+1$ valeurs donc on peut toujours découper en deux paquets de valeurs de taille supérieure à $\lfloor (n+1)/2 \rfloor$,
  - la valeur qui découpe les deux parties est insérée dans le nœud parent et on applique cette stratégie récursivement,
- si on insère dans la racine et qu'il n'y a plus de place, on crée une nouvelle racine

Arbre de recherche B+ : **Insertion**

Exemple d'Insertion : Le bloc de données comporte 3 valeurssoit l'arbre ci-dessous, on rajoute successivement les valeurs 4 et 8



Arbre de recherche B+ : **Suppression**

Suppression :

- Lorsque l'on supprime une paire (valeur, pointeur) d'une feuille, et que cette feuille reste suffisamment pleine, on en reste là,
- sinon, il y a $\lfloor (n+1)/2 \rfloor - 1$ paires, on essaie les opérations suivantes :
  - si on peut fusionner avec une feuille adjacente on le fait, on supprime une clé d'un des parents et on applique la procédure récursivement,
  - sinon, on peut traverser des paires depuis l'un des voisins.

Indexation dans les BDs

Le hachage

Le défaut de tous les index est qu'il faut les parcourir pour localiser les données auxquelles ils renvoient.

- Les techniques de hachage permettent d'éviter ce handicap,
- Elles permettent de :
  - Offrir un accès rapide à tous les tuples d'un fichier satisfaisant un même critère
  - Eviter le coût lié à la traversée d'une arborescence

Principe du hachage

- Calculer l'adresse du tuple à l'aide d'une fonction de hachage appliquée à la clé de recherche
- La sélection se fait directement en recalculant cette même fonction

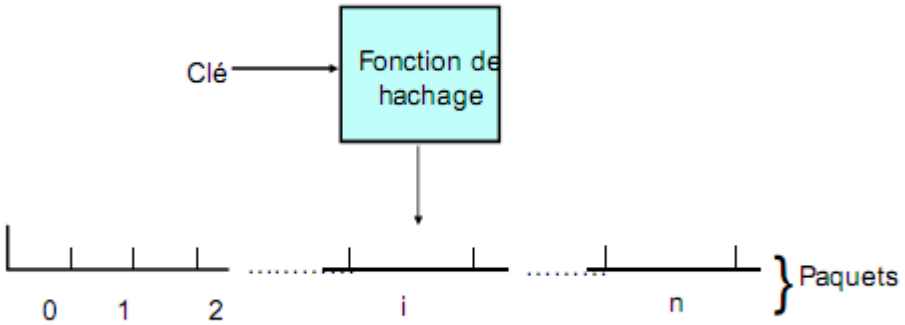
Le hachage

Fonctionnement du hachage

- Le hachage utilise une technique basée sur le calcul de l'adresse d'une information dans la BD à l'aide d'une fonction dont la variable est la valeur de la clé de tri :

$$B_i = h(K_i)$$

- Où ***K*** : ensemble de toutes les valeurs de clés de tri,
- ***B*** : ensemble de toutes les adresses des données de la BD,
- ***h*** : fonction de hachage (application de K vers B).



Index Bitmap

Un index **bitmap** est organisé sous la forme d'un tableau de bits, très pratique pour des opérations booléennes.

- tableau qui contient autant de colonnes que de valeurs possibles de la clé, et autant de lignes que la relation à indexer.
- chaque case (x; y) contient un bit qui indique si la ligne x a la valeur de clé y ; la ligne ne comporte que des 0 si la valeur de la clé vaut null.

Rowid	M	F
213	1	0
234	0	1
395	1	0
423	0	0
…	…	…

Index Bitmap

Rowid	Id_Employe	sexe	age	Id_service	…
213	1	'M'	46	null	
234	2	'F'	52	13	
395	3	'M'	28	2	
423	4	null	34	2	
…	…	…			

Table RH

Indexation dans les BDs

Index Bitmap

Exemple

```
select  *   from  Personne
where     sexe='M'   and    situation = 'Marié'
```

rowid	'M'
123	0
153	1
264	1
391	0
...	...

AND

rowid	'marié'
123	1
153	1
264	0
391	0
...	...

=

rowid	Req
123	0
153	1
264	0
391	0
...	...

Ressources utilisées

- Gestion de la mémoire et principes d'indexation dans les bases de données
  Mikaël Monet, Charles Paperman, Sylvain Salvati
- Modèles de stockage et d'indexation
  Courtesy of N. Anciaux, L. Bouganim, P. Pucheral, P. Bonnet, D. Shasha, M. J. Zaki