III- Méthodes d'index

1) Index primaire

Avec les structures de fichiers simples, quand le fichier de données devient volumineux, les opérations d'accès (recherche, insertion, ...) deviennent très inefficaces.

Les méthodes d'index permettent d'améliorer, dans une certaine mesure, les performances en gérant une structure auxiliaire (table d'index) accélérant la recherche.
Un index est (généralement) une table ordonnée contenant les couples <clé, adr > utilisée pour accélérer la recherche des enregistrements d'un fichier. Si le champ clé ne contient de valeurs en double, l'index est alors « primaire »

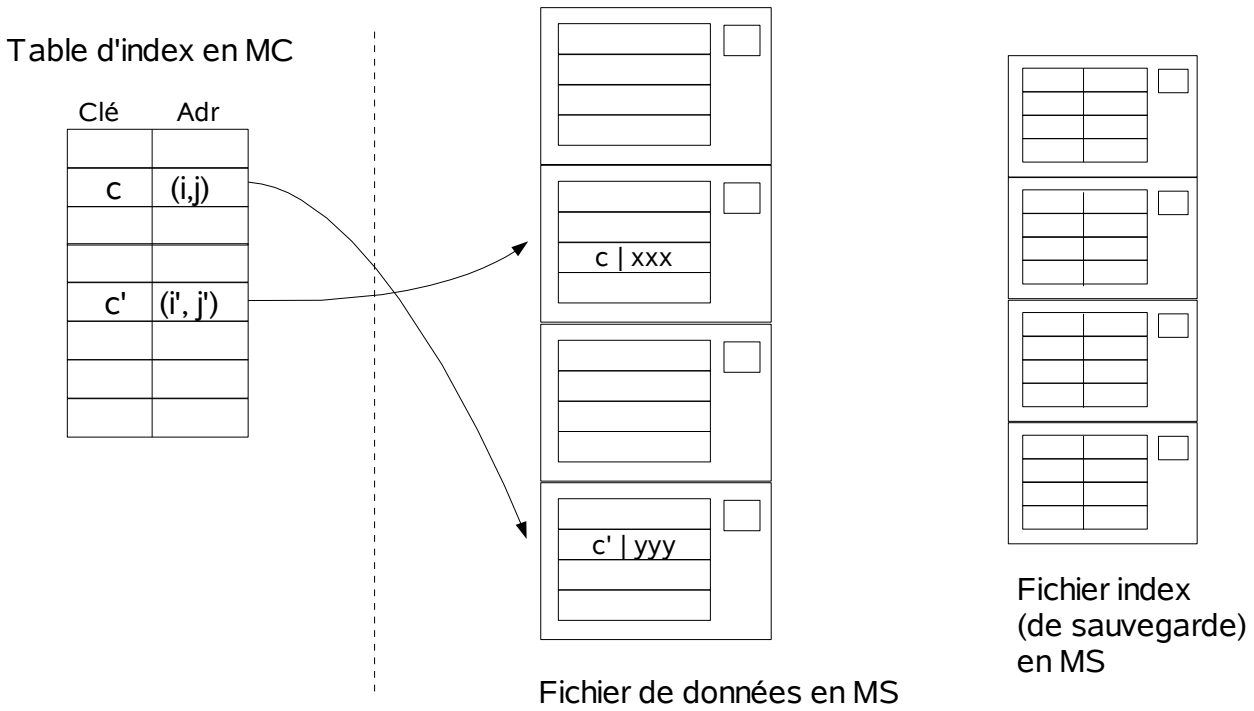
Un index est dit « dense » s'il contient toutes les clés du fichier de données. Dans ce cas le fichier n'a pas à être gardé ordonné.
Un index est dit « non dense » s'il ne contient pas toutes les clés du fichier de données (par exemple on garde une clé par bloc). Dans ce cas le fichier doit être ordonné.

Index à un niveau

Généralement, le fichier de données n'est pas ordonné
=> simplifie l'insertion et la suppression

On maintient en MC une table ordonnée, contenant toutes les clés du fichier de données (index dense). A chaque clé est alors associée l'adresse de l'enregistrement dans le fichier de données. L'adresse est un couple de nombres : <num_bloc, déplacement>

Pour ne pas avoir à reconstruire la table d'index à chaque démarrage, on sauvegarde la table dans un fichier en fin de traitement.



Les fichiers de données et d'index peuvent être de n'importe quelle structure (blocs contigus, blocs chaînés, ...). De même que les enregistrements peuvent être à format fixe ou variable (avec ou sans chevauchement).

Les opérations de base

- La recherche d'un enregistrement consiste à faire une recherche dichotomique de sa clé dans la table d'index, si elle existe, on récupère l'enregistrement à partir du fichier de donnée avec un seul accès disque.
=> coût de l'opération : 1 accès disque (au max).

- La requête à intervalle consiste à rechercher tous les enregistrements dont la clé appartient à un intervalle de valeurs donné [a,b].
1- On commence par rechercher la plus petite clé >= 'a' dans l'index (recherche dichotomique en MC).
2- Puis on continue séquentiellement dans la table jusqu'à trouver une clé > 'b'.
3- Pour chaque clé, on accède au fichier de données pour récupérer l'enregistrement.
Ce dernier point peut être amélioré si on tri les numéros de blocs trouvés avant d'accéder au fichier de données (afin de ne pas lire 2 fois le même bloc)
=> coût de l'opération en nombre d'accès : le nombre de clés vérifiant la requête (au max).

- L'insertion d'un nouvel enregistrement se fait en fin de fichier de données. Sa clé est insérée dans la table d'index (en MC) avec décalages pour garder l'ordre des clés.
=> coût de l'opération : 2 accès disques

- La suppression dans le cas d'un format variable avec chevauchement sera logique s'il n'y a pas de gestion de trous.
=> coût : 0 accès si le bit d'effacement se trouve dans la table d'index, 1 à 2 accès sinon.
Dans les autres cas, la suppression peut être physique en déplaçant par exemple le dernier enregistrement du fichiers à la place de celui effacé.
=> coût : 4 accès disques (au max)

Cas particuliers

Si le fichier de données est ordonné, on ne garde qu'une clé par bloc (par exemple la plus grande du bloc) dans la table d'index. C'est alors un index non dense (car il ne contient pas toutes les clés). Cela permet de diminuer la taille de la table d'index tout en gardant pratiquement les mêmes performances de recherche qu'une méthode d'index avec fichier non ordonné.
La requête à intervalle est beaucoup plus performantes, car dans un même bloc on peut récupérer plusieurs enregistrements vérifiant la condition de la requête.
L'insertion est par contre coûteuse, à cause des décalages dans le fichier de données.
Une solution serait alors de maintenir une zone de débordement dédiée aux enregistrements issus des décalages inter-blocs.
Les suppressions sont logiques.

On peut aussi représenter l'index en mémoire centrale sous forme d'un arbre de recherche binaire (au lieu d'une table ordonnée). L'avantage est d'éviter les décalages lors de l'insertion de nouvelles clés dans l'index. L'inconvénient est le risque du déséquilibre de l'arbre ou alors le surcout associé au maintient de l'équilibrage par des algorithmes

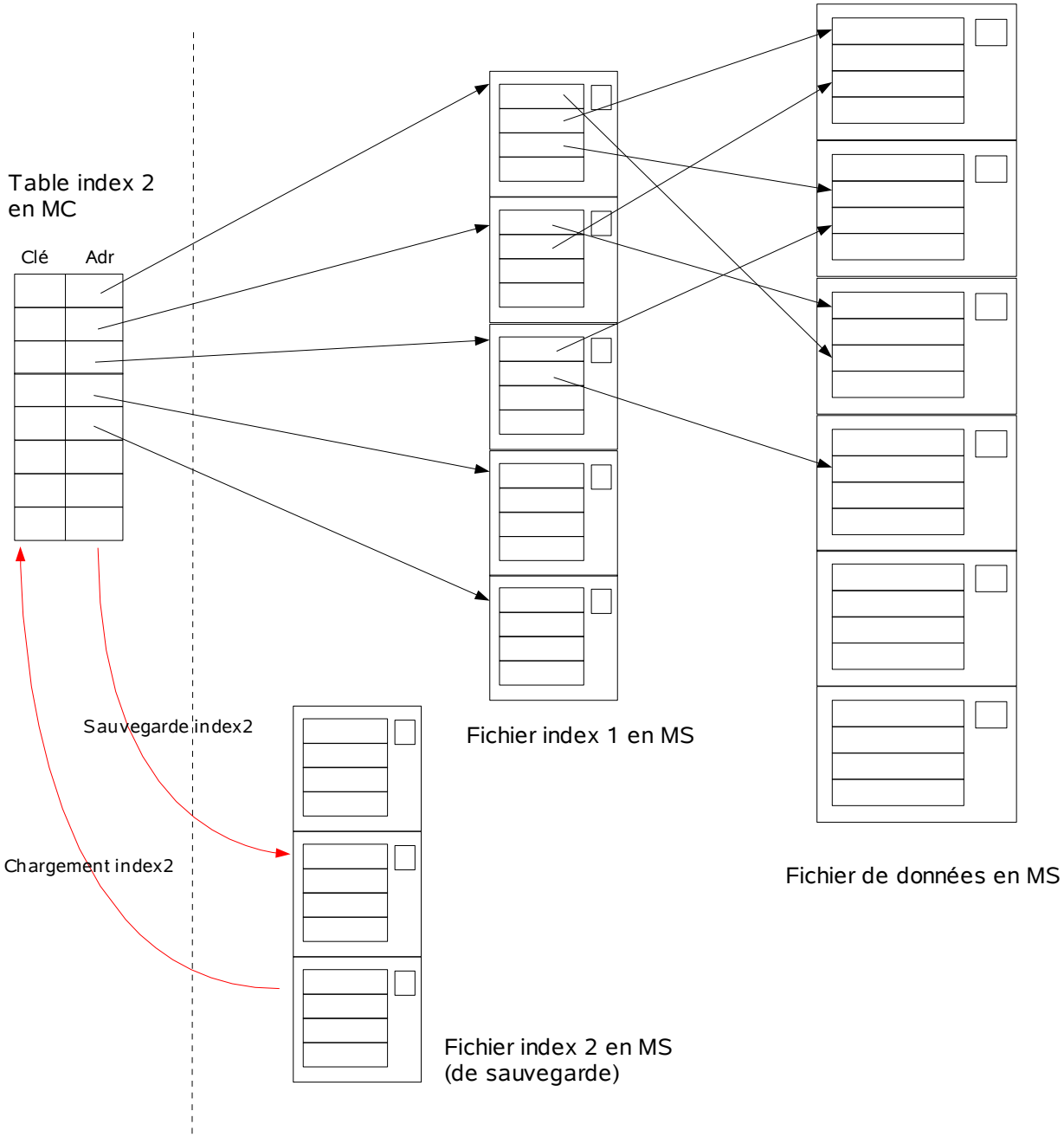
appropriés.

Index multiniveaux

Si l'index est trop grand pour résider en MC, on construit un deuxième index sur le fichier index (ordonné). Dans ce cas, on choisira une seule clé pour chaque bloc du fichier index (index non dense) pour construire le 2e index.
Si le 2e index est encore trop grand pour résider en MC, on le stocke sur disque (fichier 2e index) et on construit un 3e index en choisissant une clé par bloc du 2e fichier index.

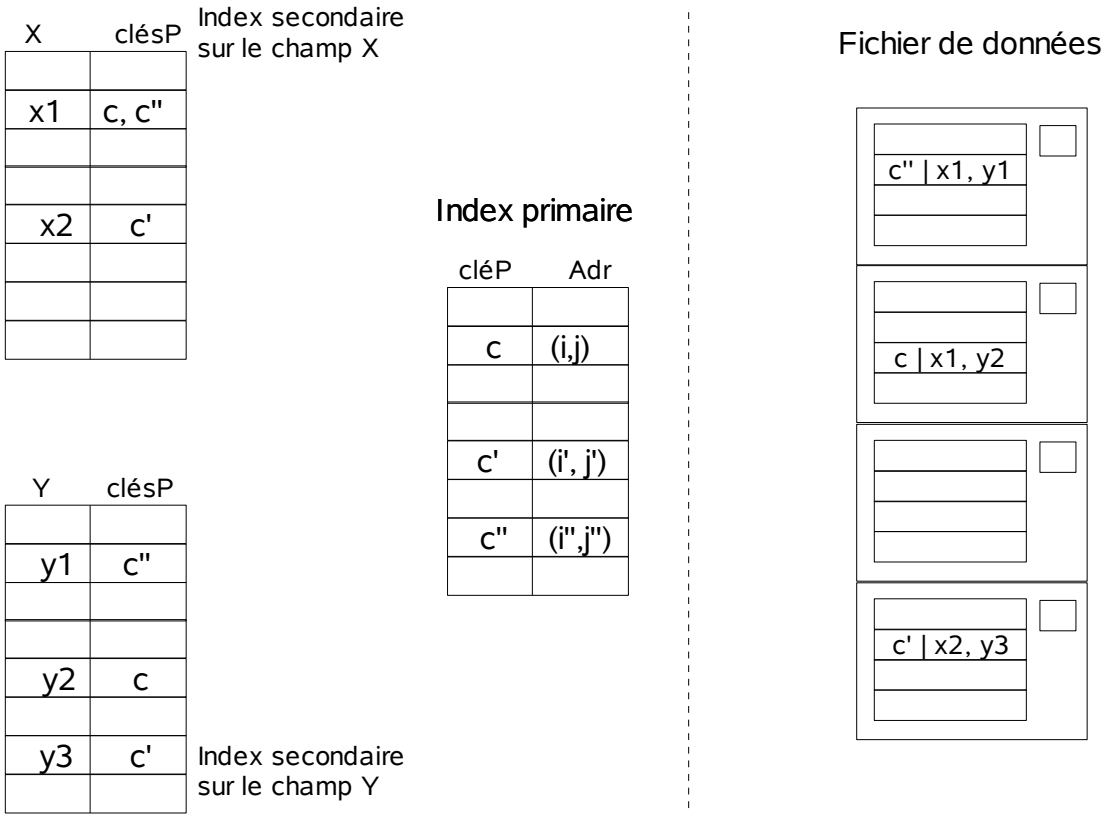
On peut répéter ce procédé au tant que nécessaire.

Dans la figure ci-dessous, un index à 2 niveaux:

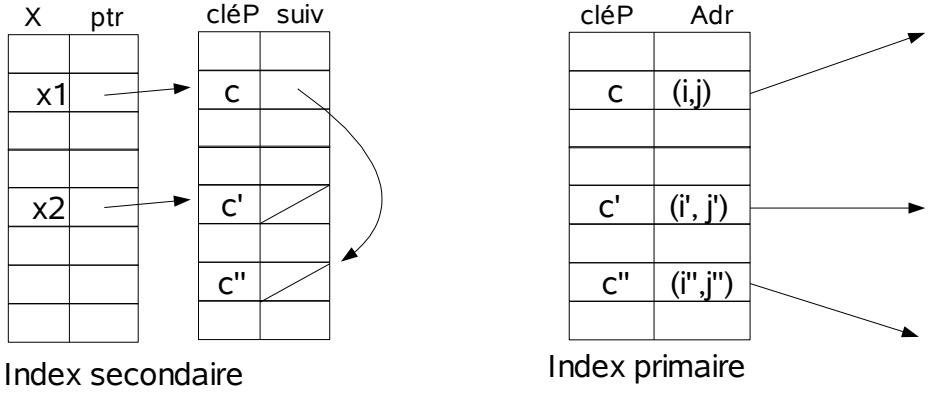


2) Index secondaire

Pour améliorer les recherches basées sur des champs non clés (appelés aussi clés secondaires), on peut construire des index secondaires sur ces champs.



Le problème avec les clés secondaires, est qu'il peut exister plusieurs enregistrements pour une valeur du champ indexé. On implémente généralement cette multiplicité à travers des listes de clés primaires



Quand on recherche les enregistrements suivant une clé secondaire (par exemple X=a), on utilise l'index secondaire sur ce champ pour récupérer la ou les clés primaires associées à la valeur cherchée (a). Pour chaque clé primaire trouvée, on utilise l'index primaire pour localiser l'enregistrement sur le fichier de données (numéro de bloc et déplacement). C'est la méthode des listes inversées.

Pour les recherches multi-critères de la forme « trouver tous les enregistrements dont la valeur de X = a ET la valeur de Y = b ET ... » avec X, Y, ... des clés secondaires, on procède comme suit :
a- En utilisant l'index secondaire X, trouver la liste Lx de clés primaires associées à la valeur de X (a).
b- Refaire la même action pour chaque clé secondaire mentionnée dans la requête...
c- Faire l'intersection des listes de clés primaires Lx, Ly, ... pour trouver les clés primaires associées avec chaque valeur de clé secondaire mentionnée dans la requête.
d- Utiliser alors l'index primaire pour retrouver les enregistrements du fichier de données

Pour insérer un enregistrement <c,x,y,...> avec c sa clé primaire et x,y,... ses clés secondaires, on procède comme suit :
a- recherche c dans l'index primaire pour vérifier qu'elle n'existe pas déjà et pour trouver l'indice ip où doit être insérée cette clé (recherche dichotomique)
b- Insérer l'enregistrement à la fin du fichier de données. Soit (i,j) son adresse
c- Insérer le couple <c, (i,j)> dans la table d'index primaire, à l'indice ip (en procédant par décalages)
d- rechercher la valeur x dans l'index secondaire X,
    si x existe, rajouter c à la liste pointée par x
    si x n'existe pas, insérer x (par décalages) dans la table X. La nouvelle entrée x, pointe une liste formé par une seule clé primaire (c).
e- refaire l'étape d) pour chaque clé secondaire restante.

Pour supprimer un enregistrement de clé primaire c, il suffit de positionner un bit d'effacement au niveau de la table d'index primaire, au niveau de l'entrée c.
Cela permet de ne pas avoir à mettre à jour tous les index secondaires.
Au niveau du fichier de données, l'enregistrement peut être supprimé physiquement si la structure du fichier le permet (comme par exemple T~OF).

Les méthodes d'index comme celles présentées dans ce chapitre, sont dédiées aux fichiers statiques (c-a-d dans les cas où le nombre d'insertions et de suppressions est relativement faible).