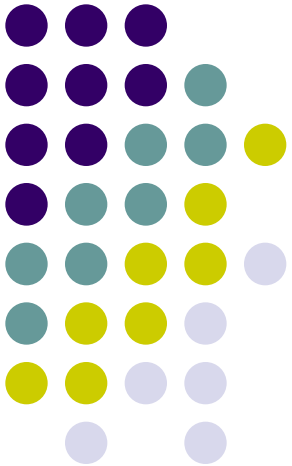
Gestion interne des bases de données

M. Thilliez[1], D.Donsez[2], N.Bennani[3]

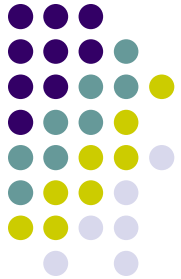
(1) UPHF

(2) Université Joseph Fourrier-Grenoble

(3) INSA Lyon

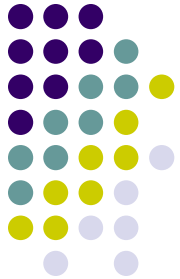


Introduction (i)



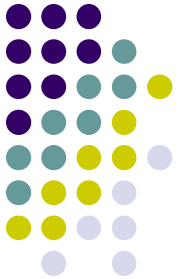
- Objectifs Initiaux des SGBD
  - Administration cohérente des données
  - Indépendance physique des données
  - Indépendance logique des données
  - Contrôle de la redondance de données
  - Manipulation des données par des non-informaticiens

Introduction (ii)



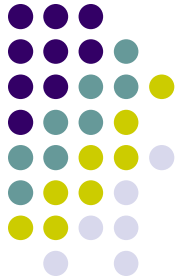
- Rappel
  - Le principe de l'indépendance physique
  - Les données sont stockées sur des supports physiques (disque dur HDD – Hard Disk Drive, disque SSD – Solid state drive)
  - Pas de traitement direct par le processeur des données sur les disques, utilisation de la RAM et de la mémoire cache.
  - Performances d'accès
    - Principale préoccupation des administrateurs de BD
    - Une demande accrue avec le développement des systèmes ouverts

Plan du cours



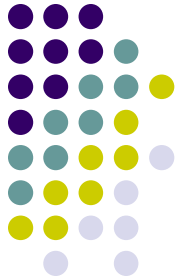
- Rappel
  - Systèmes de Gestion de Fichier
- Organisations
  - Séquentielles
  - Relatives
  - Aléatoires
  - Indexées
- Gestion des index dans Oracle

Vue interne d'une BD



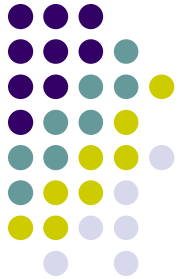
- Une base de données
  - Ensemble de fichiers
    - Fichiers de données
    - Fichiers de gestion d'accès aux données
      - Exemple:
        - Fichiers d'index
        - Répertoire de gestion de bloc disque représentant un ensemble logique,…etc
    - La métabase

Systèmes de Gestion de Fichiers (SGF)



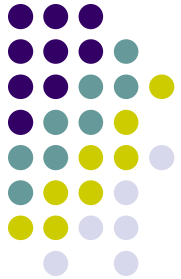
- Fichier
  - récipient de données apparentées
    pour le besoin d'un traitement
  - Caractéristiques d'un fichier
    - nom *(limité en taille)*
    - propriétaire (UID, GID sur UNIX)
    - date de création, de dernier accès, de dernière modification
    - type d'organisation, ...
- But d'un SGF
  - organiser les supports physiques en plusieurs fichiers

Systèmes de Gestion de Fichiers (SGF) [2]



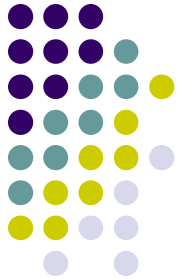
- Volume
  - unité de mémoires secondaires
- Descripteur de Fichier
  - caractéristiques du fichier
- Table de matière d'un volume
  - ensemble des descriptifs des fichiers du volume
- Uniformité des Temps d'Accès aux Données
  - Quelque soit l'emplacement des données dans un fichier, leur temps d'accès est "constant" (ce n'est pas le cas avec le SGF d'Unix)

Allocation Disque



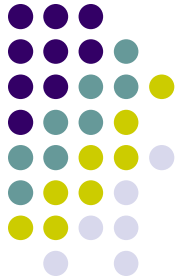
- Fichier = Structure dynamique
  - Un ficher croît au fil des ajouts
    et éventuellement décroît
  - Un fichier sur disque se compose de plusieurs morceaux répartis sur le disque.
  - Chaque morceau = 1 ou plusieurs granules d'allocation dans le disque
  - granule = N (fixe) granules physiques (secteur) "contiguës".
- Fragmentation d'un disque
  - allocations concurrentes, désallocations successives, ...
  - éparpillement des granules dans les fichiers = lectures séquentiellement logiques effectuent des "sauts physiques" dans le disque.
  - => Opération de défragmentation

Article



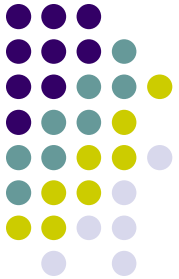
- groupe logique de données utilisées ensemble lors des traitements
  - ex: un étudiant (nom, prénom, #SS)
    dans un fichier d'étudiants
- clé d'un article
  - groupe de données identifiant l'article de façon unique dans le fichier
    - *clé primaire, clé discriminante, clé unique*
  - exemples:
    - nom et prénom
    - #SS

Enregistrement



- implante la notion d'article dans un fichier
- enregistrement de taille fixe
  - tous les enregistrements du fichier ont la même taille
- enregistrement de taille variable
- adresse d'un enregistrement
  - adresse relative :
    position de l'enr. dans un fichier
    comme s'il se composait d'un seul bloc contigu

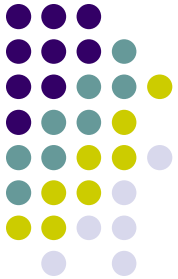
Organisations de Fichier



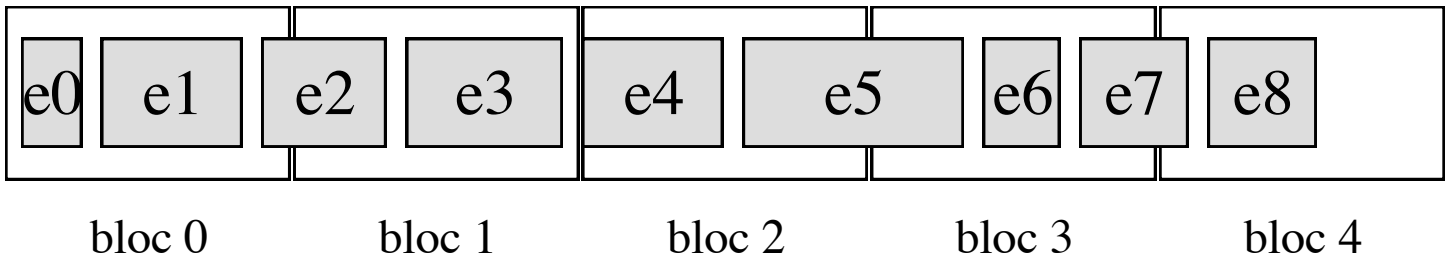
- **Organiser les enregistrements dans les fichiers en vue d'optimiser le traitement des données**

- Plusieurs types d'organisation
  - Séquentielle
  - Relative
  - Aléatoire
  - Indexée

Organisation Séquentielle



- Les enregistrements sont placés les uns à la suite des autres
  - Les suppressions créent des trous
  - Les modifications en place ne sont possibles que si :
    nouvelle longueur =< ancienne
- Chevauchement de blocs
  - occupation optimale
  - mais complique la gestion des allocations/suppressions

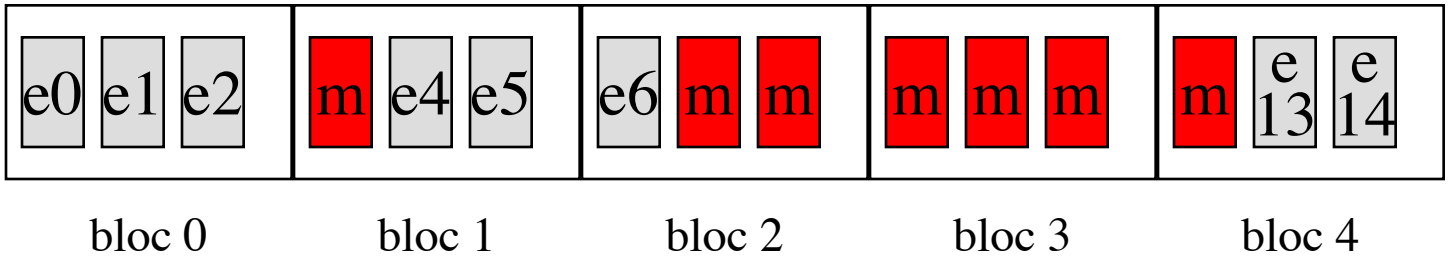


Organisation Séquentielle(2)

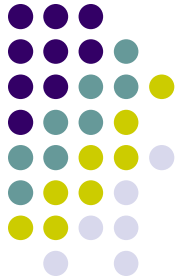
- Interface de programmation
  - Ouvrir(Nom_Fic, Mode)
  - LireEnr(Descr_Fic, Zone_Enr)
  - EcrireEnr(Descr_Fic, Zone_Enr)
  - Fermer(Descr_Fic)
- Allocations possibles :
  - Séquentielles :
    - si ajout d'un nouveau bloc devoir tout deplacer.
    - Avantages :
      - traitement séquentiel possible.
      - Traitement direct possible
  - Allocation (zone princ, zone secondaires)

Organisation Relative

- Enregistrements de longueur fixe
  - un emplacement réservé pour chaque valeur de clé possible
  - les emplacements non occupés sont marqués par une valeur de clé spéciale



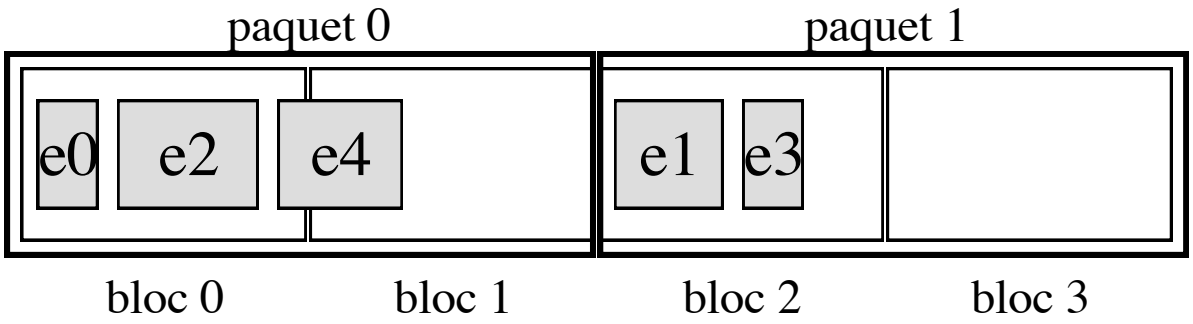
Organisation Relative(2)



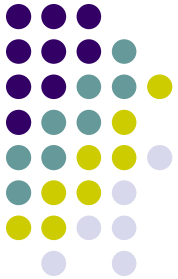
- Adressage par clé
  - adr_enr = (#bloc,#enrbloc)
  - #bloc = cle div (taille_bloc div taille_enr)
  - #enrbloc = cle mod (taille_bloc div taille_enr)
- Interface de programmation
  - LireEnr(Descr_Fic, Clé, Zone_Enr)
  - EcrireEnr(Descr_Fic, Clé, Zone_Enr)
  - EffacerEnr(Descr_Fic, Clé)
- Inconvénients
  - hypothèses trop restrictives
  - place inoccupée si faible nombre d'enr dans le fichier par rapport au nombre de valeur de clé

Organisation Aléatoire

- Le fichier est composé de P paquets (« bucket ») de taille fixe (1 à N blocs)
- Adressage en 2 étapes
  1. la clé est convertie en numéro de paquet par une fonction de hachage
  2. dans le paquet, les enregistrements ne sont pas triés (organisation séquentielle)



Organisation Aléatoire(2)

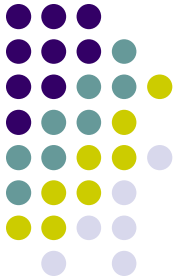


- Fonction de Hachage
  - Hash(chaine de bits) = { 0, ... , P-1 }
  - méthodes
    - Pliage de la clé
    - Conversion
    - Modulo P
    - Pseudo Aléatoire
  - but : obtenir une distribution uniforme pour saturer un paquet
  - Mauvais fonctionnement de la fonction de hachage

    ↳saturation dans un paquet

    ↳**solution :** autoriser les débordements

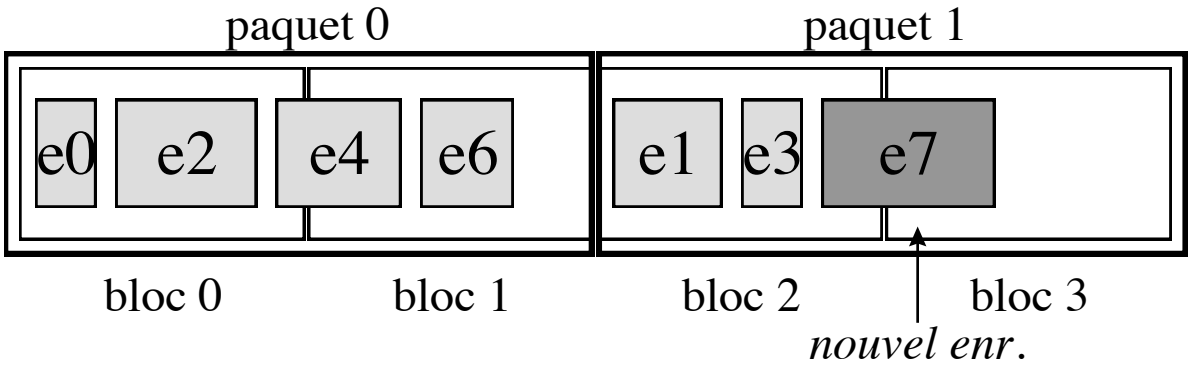
Gestions des Débordements



- Principe:
  - insertion dans un paquet en saturation
- Plusieurs stratégies
  - Adressage Ouvert
  - Zone de débordement
  - Réorganisation

Adressage Ouvert

- Principe:
  - Rechercher de la place dans le premier paquet disponible



Inconvénients :

en cas de saturation, organisation séquentielle

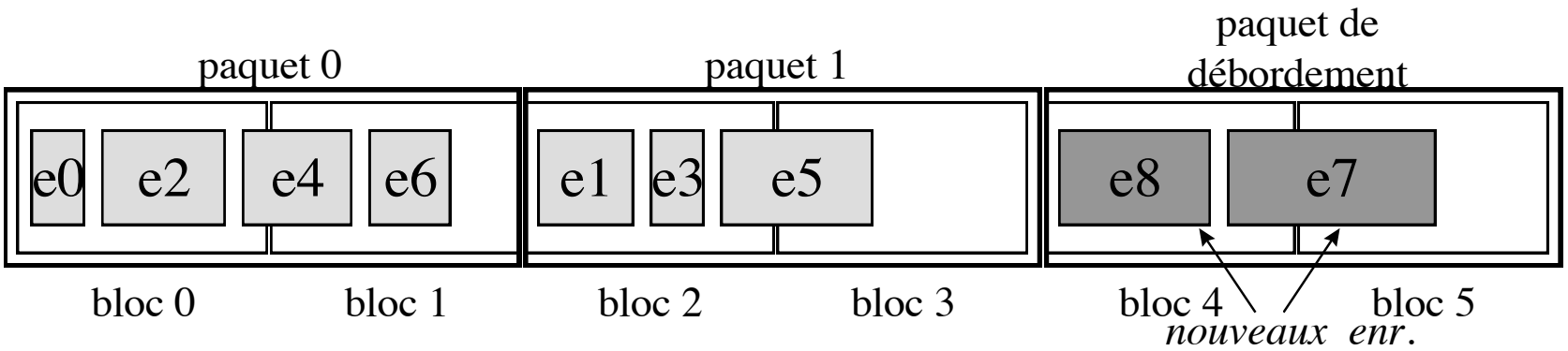
Stockage supplémentaire:

Tous les paquets ou un paquet a débordé

Chainage – zone de débordement

Principe:

Insertion dans le paquet de débordement (commun aux paquets et de taille variable)



- Inconvénient :
  - en cas de saturation de plusieurs paquets, équivalent à une organisation séquentielle.

Réorganisation

Méthode:

1. Ajout de nouveaux paquets (P' paquets dans le fichier)
2. Définir une nouvelle fonction de hachage
3. Hash'(clé) = { 0, ... , P'-1 }
4. Redistribuer tous les enr. dans les P' paquets
5. Insérer le nouvel enr.

Avantage :

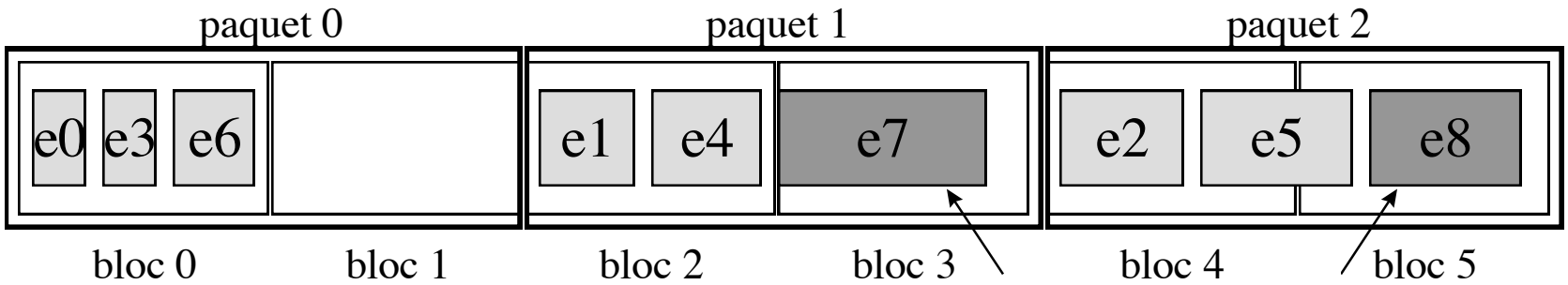
toujours une organisation aléatoire

Inconvénient :

coût élevé de la réorganisation

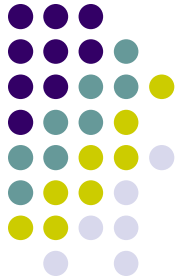
réorganisation "bloquante"

- temporisation d'une réorganisation avec un paquet de débordement de taille fixe



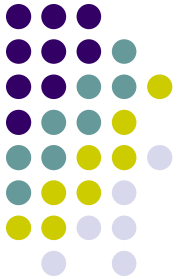
après ajout d'un paquet et redistribution, nouveaux enr.

Hachages Dynamiques



- Problème de hachage statique
  - P paquets de taille fixe
    - nécessite des réorganisations (saturation du fichier)
  - nécessite des réorganisations progressives (incrémentale)
- Hachages dynamiques
  - hachage linéaire
  - hachage extensible

Hachage Linéaire



- Etape 0:

  le fichier a P paquets Hash0(clé) = clé modulo P

- Etape 1:

  le paquet i déborde

  Le paquet 0 est éclaté en (paquet 0 + paquet P)

  le débordement de i est géré en adressage ouvert

- Etape 2:

  le paquet j déborde

  le paquet 1 est éclaté en (paquet 1 + paquet P+1)

- ...

- Etape P:

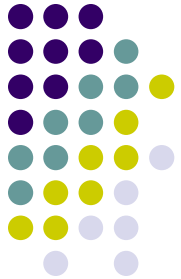
  le paquet k déborde

  le paquet P-1 est éclaté en (paquet P-1 + paquet 2P-1)

- le fichier a désormais 2P paquets

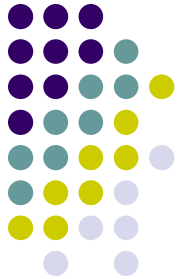
- retour à l'étape 0 avec P' = 2P

Hachage Linéaire(2)



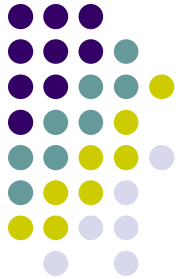
- Adressage
  - à l'étape i (le fichier a P+i paquets) ,
    - Hash(clé) = si clé mod P < i alors clé mod 2P sinon clé mod P

Hachage Linéaire(3)



- En conclusion:
  - Inconvénients:
    - Ne s ’applique qu’aux clés numériques
    - Duplique les paquets indépendamment des besoins dans un premier temps
  - Avantages:
    - Réorganisation progressive

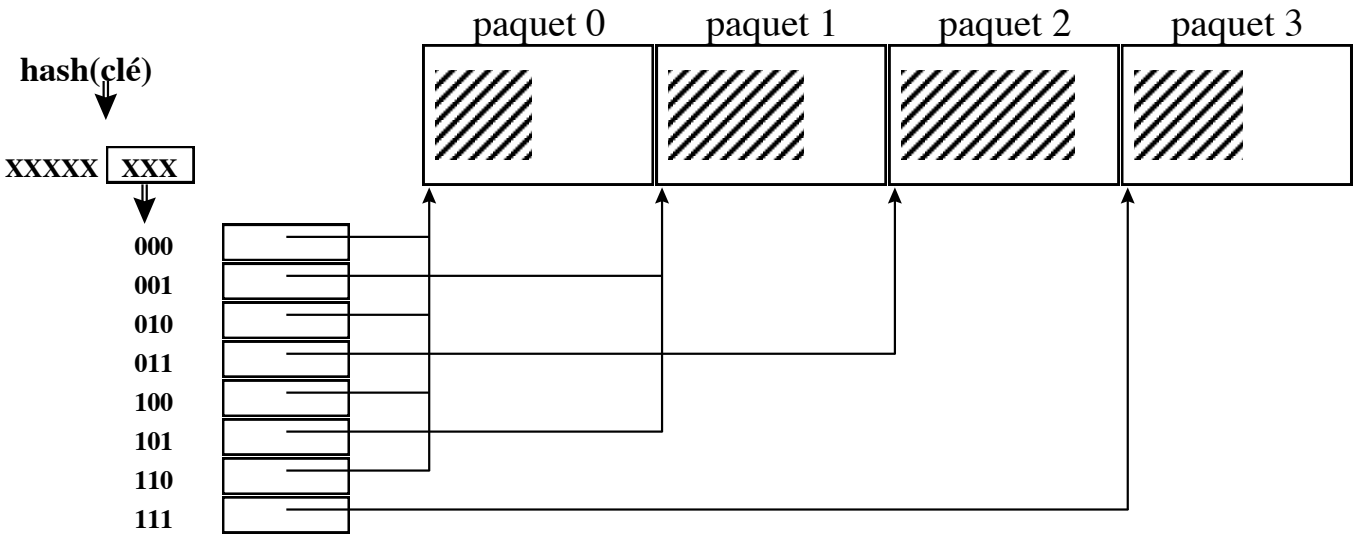
Hachage Extensible



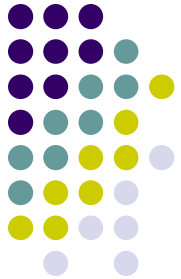
- Inconvénients du hachage linéaire
  - gestion des débordements
  - rattrapage progressif des débordements
- Idée de base
  - Gérer un répertoire des adresses relatives des paquets
  - Doubler le répertoire à chaque extension
- Inconvénient
  - Stockage du répertoire en mémoire secondaire

Hachage Extensible(2)

- Technique d'adressage
  - **Exemple : initialement 1 seul paquet, le paquet 0 (xxx)**
    - **éclatement du paquet 0 (xxx) en 0 (xx0) et 1 (xx1)**
    - **éclatement du paquet 1 (xx1) en 1 (x01) et 2 (x11)**
    - **éclatement du paquet 2 (x11) en 2 (011) et 3 (111)**

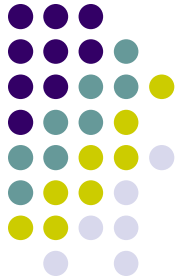


Organisations Indexées



- Objectifs
  - Accès rapide par clé
  - Accès séquentiel Trié ou Non
- Moyen
  - "Table" permettant d'associer à chaque clé l'adresse relative de l'enregistrement correspondant.

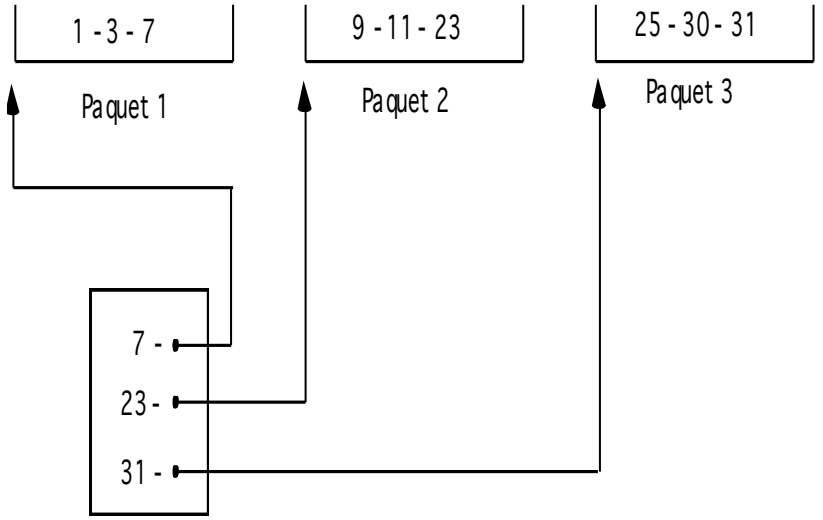
Classification des indexs



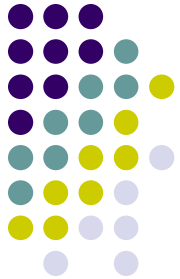
- Index Dense / Index Non Dense
  - l'index dense contient toutes les valeurs de clé trouvées dans les enregistrements
  - l'index non dense n'est possible que si le fichier est trié. L'index contient la plus grande valeur du bloc seulement
- Index primaire /secondaire :
  - Index primaire
    - Sur un attribut (ou un groupe d'attributs) clé
    - Chaque valeur de clé est unique
  - Index secondaire :
    - Sur un attribut (ou un groupe d'attributs) non clé
    - Une valeur d'attributs peut correspondre à plusieurs enregistrements

Classification des indexs(2)

- Index lexicographique :
  - La clé est une chaîne de caractères (CHAR(), VARCHAR()) parfois longue.
  - Les nœuds intermédiaires ne comportent que le début de la chaîne de la clé
- Index hiérarchisés
  - Chaque index fils possède un index père permettant de rechercher plus rapidement une entrée dans l'index fils
  - Utilité:
    - Gestion rapide des gros fichiers
-



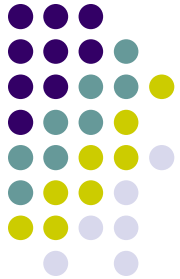
Les Arbres de Bayer (B-Tree)



- Problème :
  - trouver une structure disque dynamique qui permette d'implanter la "table d'index"

    ↳structure d'arbre équilibré ↳Arbre B
- Propriété
  - les arbres restent équilibrés quelques soient:
    - les insertions
    - les suppressions.

Les B-tree: définition



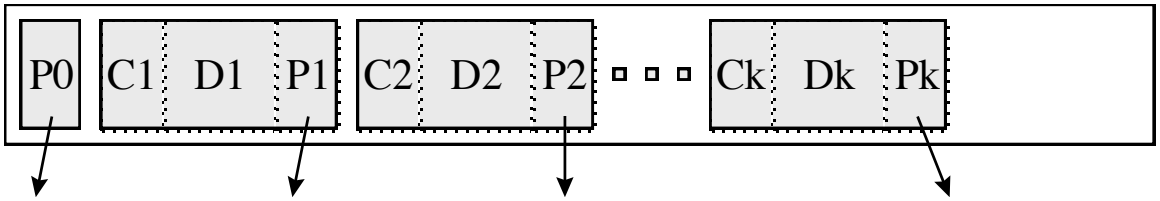
- Définition : Arbre-B d'ordre m
  - **arborescence telle que :**
    1. chaque chemin depuis la racine à une feuille est de même longueur H (hauteur)
    2. chaque nœud contient K clés triés avec K tel que
       $M \leq K \leq 2M$
       $0 \leq K \leq 2M$ pour la racine
  - **un noeud est:**

    soit Terminal

    soit possède K+1 fils tels que les clés du Ième fils ont des valeurs comprises entre les valeurs des clés (I-1) et I du père

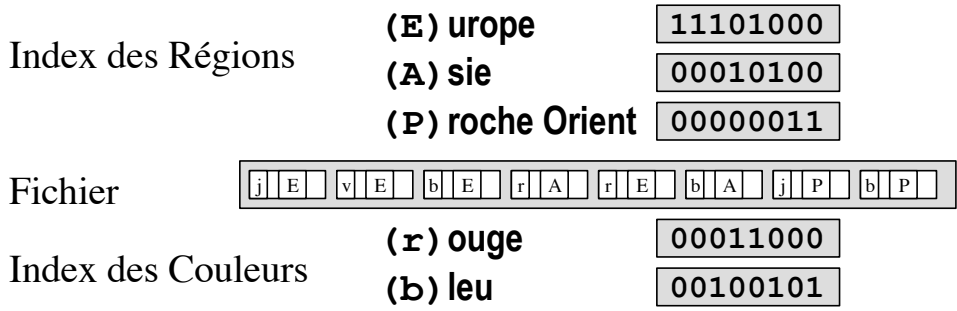
Les Arbres B+

- Définition : Arbre-B+
  - arbre B dans lequel les clés des noeuds ascendants sont répétés dans chaque noeud descendant et on chaîne les noeuds feuilles pour permettre un accès rapide en séquentiel trié.
- Implantation
  - chaque noeud est contenu dans un bloc
    - Pi : pointeur disque vers un fils
    - Ci: valeur de clé
    - Di: valeur externe



Index Bitmap

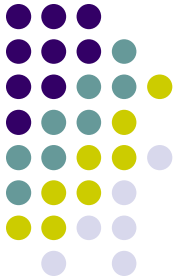
- Un vecteur de bits par valeur ou groupe de Valeur ou sur un prédicat
- Opérations logiques OR, AND, NOT



E	11101000
A	00010100
Asie OR Europe	11111100

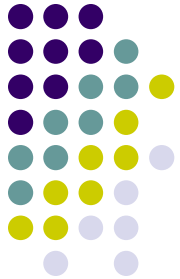
E	11101000
Not (R OR B)	11000010
Jaune AND Europe	11000000

Index Bitmap (2)



- Usage des index bitmap
  - DataWarehouses (Entrepôts de Données)
  - Fichier à indexer: jusqu'à plusieurs milliards d'enregistrements soit quelques TeraOctets
  - Enregistrements de taille fixe (Accès aléatoire dans le fichier), pas de modification, Seulement des ajouts (append only)
  - L'index est très compact , la recherche très rapide

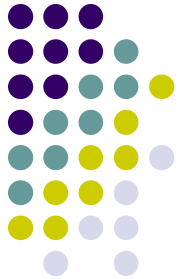
Autres types d’ Index



- Index Multi-Champ (ou Multi-Attribut)
  - La combinaison de plusieurs valeurs de champs permet de retrouver le ou les enregistrements
- Index sur Champ (Attribut,Colonne) Virtuel
  - La valeur utilisée pour l'indexation n'est pas la valeur d'un champ mais le résultat du calcul d'une fonction sur un ou plusieurs champs.
- Index de Jointure (Join Indice)
  - La valeur utilisée pour l'indexation n'est pas la valeur d'un champ de l'enregistrement mais celle d'un champ d'un autre enregistrement atteignable par une clé étrangère.
  - Exemple : Soient 2 fichiers
    - Client(Ncli, Nom, Ville)
    - Vente(Ncli, Nprod, Qte, Date)

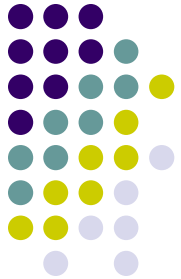
    - Vente peut être indexé sur la Ville du Client

Index dans Oracle



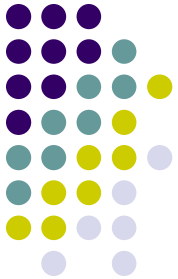
- Types d'index
  - Arbre B+ (implicite)
  - Arbre B
  - Table de hachage
- Création d'un index
  - Implicite avec PRIMARY KEY / UNIQUE
    - **Vérification de la contrainte FOREIGN KEY**
  - Explicite
    - **CREATE INDEX nom_index ON nom_table(col1[,col2])**
- Suppression
  - DROP

Index dans Oracle



- Remarque:
  - La gestion d'index ne fait pas partie du standard SQL

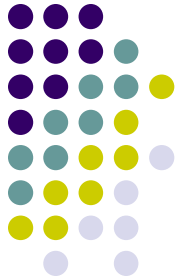
Création d'un arbre B



- Arbre B plaçant
- Sur la clé
- Ordre de création

  CREATE TABLE <nom de table>
  (champClé type (...),
  ...
  PRIMARY KEY (champClé),
  ORGANIZATION INDEX)

Index dans SQL



- Performances
  - Amélioration pour les requêtes portant sur une partie des lignes de la table
  - Coût supplémentaire lors des MAJ
- Critères de choix des colonnes à indexer
  - colonne fréquemment utilisée dans une clause WHERE
  - colonne fréquemment utilisée dans une jointure
  - colonne ayant une bonne « sélectivité », c'est à dire colonne dont peu d'enregistrements ont la même valeur
  - colonne rarement modifiée
  - colonne n'apparaissant pas dans une clause WHERE avec des fonctions ou des opérateurs
- Index « composé » ou multi-colonne
  - Augmenter la « sélectivité ».
- Index secondaire