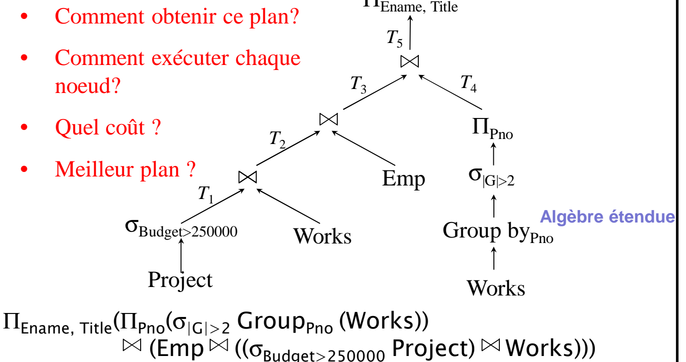
3I009 Licence d'informatique

Cours 4 – Optimisation de requêtes

Stephane.gancarski@lip6.fr

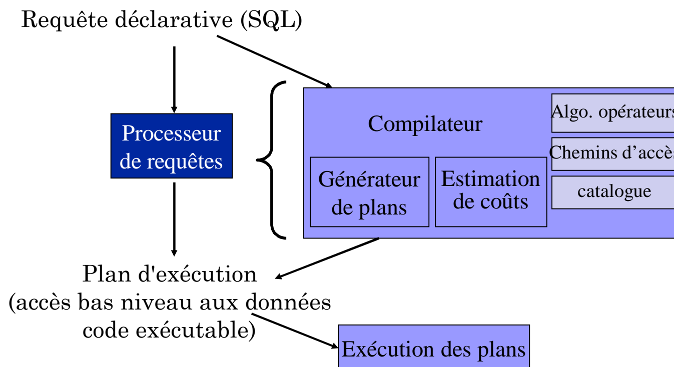
Plan d'exécution

- Comment obtenir ce plan?
- Comment exécuter chaque noeud?
- Quel coût ?
- Meilleur plan ?



$\Pi_{Ename, Title}(\Pi_{Pno}(\sigma_{|G|>2}\ Group_{Pno}\ (Works)) \bowtie (Emp \bowtie ((\sigma_{Budget>250000}\ Project) \bowtie Works)))$

Traitement des requêtes



Cours 4 - Optimisation de requêtes

1. Organisation des données, chemins d'accès
2. Implémentation des opérateurs relationnels
3. Restructuration de la requête
4. Coût des opérations
5. Optimisation du coût
6. Espace de recherche
7. Stratégie de recherche

Stockage des données

- Les données sont stockées en mémoire non volatile
  - Disque magnétique, flash (carte SD, disque SSD), bande magnétique,
- Gestion de l'espace disque
  - L'unité de stockage est : la page
  - La taille d'1 page est fixe pour un SGBD (souvent 8Ko, parfois plus)
- 2 opérations élémentaires pour accéder au données stockées:
  - lire une page, écrire une page
- Le coût d'une opération SQL dépend principalement du nombre de pages lues et/ou écrites.
  - Coût E/S >> Coût calcul en mémoire
  - Coût dépend donc fortement de la façon dont les données sont organisées sur le disque → modèle de coût complexe.
- Gestion de l'espace en mémoire centrale
  - Réalisée par le SGBD (gestionnaire de tampon)
  - Gestion dédiée plus efficace qu'un OS généraliste.

Organisation des données

- Un enregistrement représente une donnée pouvant être stockée.
  - ex. une ligne ou une colonne d'une table
- Les enregistrements sont stockées dans les pages d'un fichier
- Un enregistrement a un identificateur unique servant d'adresse pour le localiser
  - (idFichier + idpage + offset ⇔ rowid dans Oracle).
  - Le gestionnaire de fichier peut accéder directement à la page sur laquelle se trouve un enregistrement grâce son adresse.
- La façon d'organiser les enregistrements dans un fichier a un impact important sur les performances.
  - Elle dépend du type de requêtes. Ex. OLTP (ligne) vs. OLAP (colonne).
  - Elle dépend aussi du type de mémoire. Ex. Flash très lent écriture.
  - Ce cours : stockage sur disque, OLTP
- Un SGBD offre en général plusieurs méthodes d'accès.
  - L'administrateur de la base détermine la méthode d'accès la plus adéquate

Organisation séquentielle

- Non trié :
  - Très facile à maintenir en mise à jour
  - Parcourir toutes les pages quelque soit la requête
- Trié :
  - Un peu plus difficile à maintenir
  - Parcours raccourci car on peut s'arrêter dès qu'on a les données cherchées

En BD, il y a presque toujours un compromis à faire entre lecture et écriture

Organisations Indexées

- Objectifs
  - Accès rapide à partir d'une clé de recherche
  - Accès séquentiel trié ou non
- Moyens
  - Utilisation d'index permettant la recherche de l'adresse de l'enregistrement à partir d'une clé de recherche
- Exemple
  - Dans une bibliothèque, rechercher des ouvrages par thème, par auteur ou par titre.
  - Dans un livre, rechercher les paragraphes contenant tel mot.

Entrée d'un index

- On appelle une entrée la structure qui associe une clé de recherche avec l'adresse des enregistrement concernés
  - Adresse : localisation d'un enregistrement
- Trois alternatives pour la structure d'une entrée
  1. Entrée d'index contient les données
     - localisation directe: on n'utilise pas l'adresse
  2. Entrée d'index contient (k, ptr)
     - Pas plus d'un enregistrement par valeur
  3. Entrée d'index contient (k, liste de ptr)
     - on peut avoir plusieurs enregistrements par valeur

Index non plaçant

- Les index **non plaçants** sont dit secondaires
- Index = structure auxiliaire en plus des données
- Permet d'indexer des données quelle que soit la façon dont elle sont stockées
  - Donnés stockées sans être triées
  - Données triées selon un attribut **autre** que celui indexé
- Définir un index non plaçant en SQL
  - create index *NOM* on *TABLE*(*ATTRIBUTS*);
    - create index IndexAge on Personne(âge)

Index plaçant

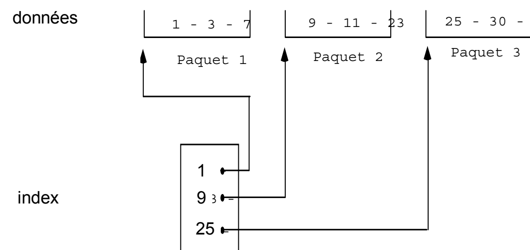
- Le stockage des données est organisé par l'index.
- Les enregistrements ayant la même valeur de clé sont juxtaposés
  - Stockage contigus dans un paquet et dans les paquets contigus suivants si nécessaire
- Définir l'organisation des données lors de la création de la table.
  - create table...organization index : données **triées** selon la clé primaire
  - create cluster... : données regroupées par valeur d'un attribut
- Un entrée contient les données (*cf.* alternative 1)
- Evidemment, pas plus d'un index plaçant par table
  - Appelé index principal

Index plaçant non dense

- Concerne seulement les index plaçants
  - Les données doivent être stockées triées
- Objectif: obtenir un index occupant moins de place
- Méthode: enlever des entrées. Ne garder que les entrées nécessaires pour atteindre le bloc (i.e. la page) de données contenant les enregistrements recherchés
  - Garder l'entrée ayant la plus petite (ou la plus grande) clé de chaque page.
  - Ne pas indexer 2 fois la même clé dans 2 pages consécutives
- Inconvénient: toutes les valeurs de l'attribut indexé ne sont pas dans l'index. Cf diapo (index couvrant une requête)

- Rmq: Un index contenant **toutes** les clés est dit **dense**

Exemple d'index plaçant non dense



Index unique

- Un index est dit unique si l'attribut indexé satisfait une contrainte d'unicité
  - Contrainte d'intégrité:
    - attribut (ou liste d'attributs) déclaré(e) comme étant : unique ou primary key
  - Une entrée a la forme (clé, ptr)
  - *cf.* alternative 2
- Sinon : cas général d'un index **non** unique
  - Une entrée a la forme (clé, **liste** de ptr)
  - *cf.* alternative 3

Accès aux données par un index

- Sert pour évaluer une sélection
  - une égalité: prénom = 'Alice'
  - l'accès est dit 'ciblé' si l'attribut est unique
  - un intervalle : age between 7 and 77
  - une inégalité : age > 18 <, >, ≤, ≥
  - une comparaison de préfixe : prénom like 'Ch%'
    - Rmq : un index ne permet **pas** d'évaluer une comparaison de suffixe. Exple prénom like '%ne'
  - Rmq: si les entrées de l'index ne sont pas triées (cas d'une table de hachage), seule l'égalité est possible de manière efficace

Index couvrant une requête

- Un index (non plaçant) couvre une requête s'il est possible d'évaluer la requête sans lire les données
- Tous les attributs mentionnés dans la requête doivent être indexés
- Index couvrant une sélection
  - Pour chaque prédicat *p* de la clause *where*, il faut un index capable d'évaluer *p*.
- Index couvrant une projection
  - Pour chaque attribut de la clause *select*, il faut un index **dense** (i.e, contenant toutes les valeurs de l'attribut projeté)
- Avantage
  - Evite de lire les données, évaluation plus rapide d'une requête
- Concerne seulement les index non plaçants
  - Un index plaçant contenant les données, elles sont forcément lues.

Index composé

- Clé composée considérée comme une clé simple formée de la concaténation des attributs
- Sélection par préfixe de la clé composée
  - Clé composée (a1, a2, a3, ..., an)
  - Il existe n préfixes : (a1), (a1,a2) , ..., (a1,a2, ...,an)
    - Rmq: (a2,a3) n'est pas un préfixe
- Index composé utilisable pour une requête
  - On appelle (p1, p2, ...p_m) les attributs mentionnés dans le prédicat de sélection
  - (p1, p2, ...p_m) doit être un préfixe de la clé composée
  - Prédicat d'**égalité** pour tous les attributs p1 à p_{m-1}
  - Egalité, inégalité ou comparaison de préfixe pour le dernier attribut p_m

Index composé: exemple

- Exemple : create index I1 on Personne(âge, ville)
- Utilisable pour les requêtes : Select * from Personne ...
  - Where âge > 18
  - Where âge =18 and ville = 'Paris'
  - Where âge =18 and ville like 'M%'
- Inutilisable pour :
  - Where ville= 'Paris'
  - Where âge > 18 and ville = 'Paris'

Choix entre un accès séquentiel ou un accès par index

- Définir un ou plusieurs index
- Poser des requêtes. Le SGBD utilise les index existants
  - s'il estime que c'est plus rapide que le parcours séquentiel des données.
  - Décision basée sur des règles heuristiques ou sur une estimation de la durée de la requête (voir TME)
- L'utilisateur peut forcer/interdire le choix d'un index
    - **Select** *
    - From Personne
    - Where age < 18
  - Devient
    - **Select /*+ index(personne IndexAge) */** *
    - From Personne
    - Where age < 18
  - Syntaxe d'une directive:
    - index(*TABLE INDEX*)
    - no_index(*TABLE INDEX*)

Index hiérarchisé

- Lorsque le nombre d'entrées de l'index est très grand
- L'ensemble des entrées d'un index peuvent, à leur tour, être indexées. Cela forme un index hiérarchisé en plusieurs niveaux
  - Le niveau le plus bas est l'index des données
  - Le niveau n est l'index du niveau n+1
  - Intéressant pour gérer efficacement de gros fichiers
  - Le plus connu : arbre B+ (+ de détail au prochain cours)

Organisation par hachage

- Les fichiers sont placés dans des paquets en fonction d'une clé
- On applique une *fonction de hachage* sur la clé d'un n-uplet, ce qui détermine l'adresse du paquet où stocker le n-uplet
- On peut rajouter une indirection *: table de hachage*
- Efficace pour des accès par égalité, pas adapté aux requêtes par intervalle (données non triées)
- La fonction de hachage doit bien répartir les données dans les paquets
- Hachage statique vs hachage dynamique (+ de détail au prochain cours)

Implémentation des opérateurs

Rappel: accès disque >> accès mémoire (négligeable)

Coûts n'incluent pas écriture temporaire éventuelle

R contient *n* pages disques

- Sélection sur égalité, sur intervalle
- Projection
- Jointure

Sélection

- Sélection sur égalité
  - parcours séquentiel (scan)
    - le nombre d'accès disques est en $O(n)$
  - Parcours (scan) avec index
    - index B^+ : $O(\log_k(n))$ /* hauteur de l'arbre + un accès par nuplet (cf TME)
    - hachage : O(1) /* statique en supposant une bonne répartition. O(2) hachage dynamique
- Sélection sur intervalle
  - parcours séquentiel (scan) : idem
  - Parcours (scan) avec index
    - index B^+ : $O(\log_k(n) + M)$) + un accès par nuplet M nombre de pages contenant des clés correspondant à l'intervalle
    - hachage : O(X) où X est le nombre de valeur dans l'intervalle

Implémentation des opérateurs

- Projection
  - sans élimination des doubles – $O(n)$
  - avec élimination des doubles
    - en triant – $O(2n\log_k(n))$
    - en hachant – $O(n+2t)$ où t est le nombre de pages du fichier haché après proj. et avant élimination
      - La fonction de hachage doit être choisie pour que, à chaque entrée de la table de hachage corresponde un nombre de pages assez petit pour tenir en mémoire
      - t vaut n si uniquement élimination des doubles, t < n si projection et élimination en même temps (les nuplets sont plus petits)

Implémentation des opérateurs

- Jointure (R sur n pages, S sur m pages)
  - boucle imbriquée (nested loop): $T = R \bowtie S$

```
foreach tuple r∈R do   /* foreach page de R
  foreach tuple s∈S do
    if r==s then T = T + <r,s>
```

    - O($n+n*m$)
  - amélioration possible pour réduire les accès disques
    - boucles imbriquées par pages ou blocs : permet de joindre chaque n-uplet (page) de S avec non plus un seul n-uplet (page) de R, mais avec tous (on suppose p) ceux qui tiennent en MC (p+1)
    - O(n+m*n/p)
    - Cas particulier si R tient en mémoire, n/p = 1, O(n+m)

Implémentation des opérateurs

- Jointure
  - boucle imbriquée et index sur attribut de jointure de S (cas typique : jointure sur clé étrangère)

```
foreach tuple r∈R do
  accès aux tuples s∈S par index (ou hachage)
  foreach tuple s do
    T = T + <r,s>  /* O(n+M),M=card(R)*k(coût index)
```

  - tri-fusion
    - trier *R* et *S* sur l'attribut de jointure : tri externe O(2nLog$_k$(n))
    - fusionner les relations triées : O(n+m)
    - Peut être optimisé en commençant la fusion avant la fin complète du tri
  - hachage
    - hacher *R* et *S* avec la même fonction de hachage : O(n+m) L + O(n+m) E
    - pour chaque paquet i de *R* et i de *S*, trouver les tuples où *r=s* : O(n+m) L

Tri externe

- Algo
  - Trier des paquets de k pages tenant en mémoire disponible

    `n/k paquets, 2n E/S`
  - Charger les premières pages de chaque paquet et trier
    - Dès qu'une page est vide, charger la suivante du même paquet
    - On obtient des paquets de k^2 pages triés

    `n/k² paquets, 2n E/S`
  - Continuer jusqu'à obtenir un paquet de $k^s >= n$ pages
- Coût total
  - A chaque étape on lit et écrit toutes les données : `2n E/S`
  - Nombre d'étape s, tel que k^s=n : $s = \log_k(n)$
  - Soit en tout 2nlog$_k$(n)
  - Nb : si tri pour fusion, pas besoin de faire la dernière étape, on fait la fusion directement avec l'autre relation

Optimisation

- Elaborer des plans
  - arbre algébrique, restructuration, ordre d'évaluation
- Estimer leurs coûts
  - fonctions de coût
    - en terme de temps d'exécution
    - coût I/O + coût CPU
    - poids très différents
      - par ex. coût I/O = 1000 * coût CPU
- Choisir le meilleur plan
  - Espace de recherche : ensemble des expressions algébriques équivalentes pour une même requête
  - algorithmes de recherche:
    - parcourir l'espace de recherche
    - algorithmes d'optimisation combinatoire

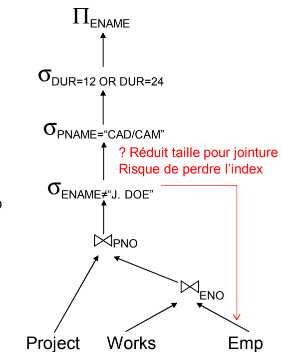
Restructuration

- Objectif : choisir l'ordre d'exécution des opérations algébriques (élaboration du plan logique).
- Conversion en arbre algébrique
- Transformation de l'arbre (optimisation)
  - règles de transformation (équivalence algébriques),
  - estimation du coût des opérations en fonction de la taille
  - Estimation du résultat intermédiaire (taille et ordre?)
  - En déduire l'ordre des jointures

Restructuration

- Conversion en arbre algébrique
- Exemple

```
SELECT  Ename
FROM   Emp, Works, Project
WHERE  Emp.Eno=Works.Eno
AND  Works.Pno=Project.Pno
AND    Ename NOT='J.Doe'
AND    Pname = 'CAD/CAM'
AND    (Dur=12 OR Dur=24)
```



Calcul du coût d'un plan

- La fonction de coût donne les temps I/O et CPU
  - nombre d'instructions et d'accès disques
- Estimation de la taille du résultat de chaque noeud
  - Permet d'estimer le coût de l'opération suivante
  - sélectivité des opérations – "facteur de réduction"
  - propagation d'erreur possible
- Estimation du coût d'exécution de chaque noeud de l'arbre algébrique
  - utilisation de pipelines ou de relations temporaires importante
    - Pipeline : les tuples sont passés directement à l'opérateur suivant.
      - Pas de relations intermédiaires (petites mémoires, ex. carte à puce).
      - Permet de paralléliser (BD réparties, parallèle)
      - Intéressant même pour cas simples : $\sigma_{F \wedge F'}(R)$, index sur F' $\rightarrow \sigma_F(\sigma_{F'}(R))$
    - Relation temporaire : permet de trier mais coût de l'écriture

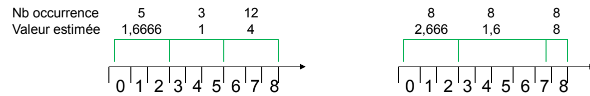
Statistiques

- Relation
  - cardinalité : card(R)
  - taille d'un tuple : largeur de R
  - fraction de tuples participant une jointure / attribut
  - …
- Attribut
  - cardinalité du domaine
  - nombre de valeurs distinctes distinct(A,R) = $\Pi_A(R)$
  - Valeur max, valeur min
- Hypothèses
  - indépendance entre différentes valeurs d'attributs
  - distribution uniforme des valeurs d'attribut dans leur domaine
  - Sinon, il faut maintenir des histogrammes
    - Equilarge : plages de valeurs de même taille
    - Equiprofond: plages de valeurs contenant le même nombre d'occurrence
    - Equiprofond meilleur pour les valeurs fréquentes (plus précis) voir transparent suivant
- Stockage :
  - Les statistiques sont des métadonnées, stockées sous forme relationnelle (cf. TME)
  - Rafraîchies périodiquement, pas à chaque fois.

Le fameux compromis L/E

Histogramme

Select * from R where A = 8



Equilarge
Valeur estimée 4
Valeur réelle dans [0,12]
(faire l'hypothèse d'uniformité dans l'intervalle)

Equiprofond
Valeur estimée 8
Valeur réelle 8

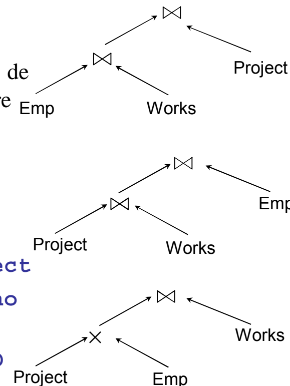
Espace de recherche

- Caractérisé par les plans "équivalents" pour une même requête
  - ceux qui donnent le même résultat
  - générés en appliquant les règles de transformation vues précédement
- Le coût de chaque plan est en général différent
- L'ordre des jointures est important

Arbres de jointures

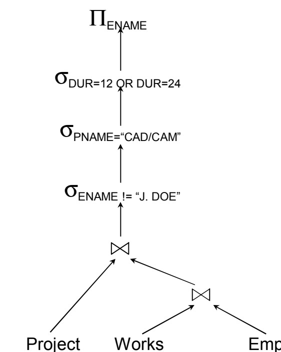
- Avec *N* relations, il y a O(*N*!) arbres de jointures équivalents qui peuvent être obtenus en appliquant les règles de *commutativité* et d'*associativité*

```
SELECT  Ename,Resp
 FROM   Emp, Works,Project
 WHERE  Emp.Eno=Works.Eno
 AND
    Works.PNO=Project.PNO
```

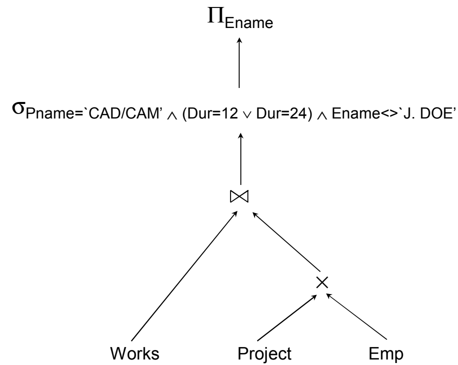


Exemple

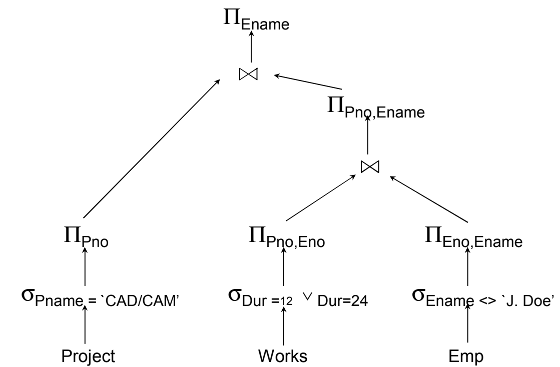
```
SELECT Ename
FROM Project p, Works w,
     Emp e
WHERE w.Eno=e.Eno
AND   w.Pno=p.Pno
AND   Ename<>`J. Doe'
AND   p.Pname=`CAD/CAM'
AND   (Dur=12 OR Dur=24)
```



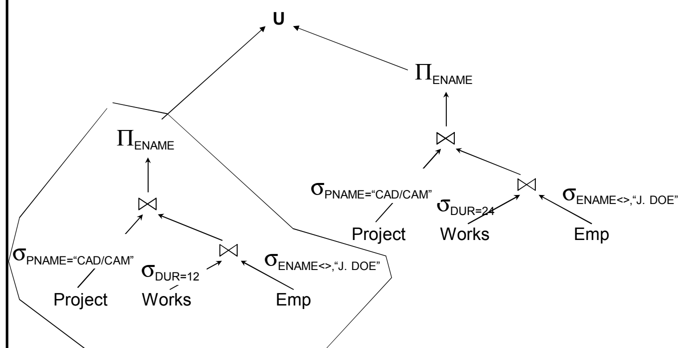
Requête équivalente



Autre requête équivalente



Encore une requête équivalente

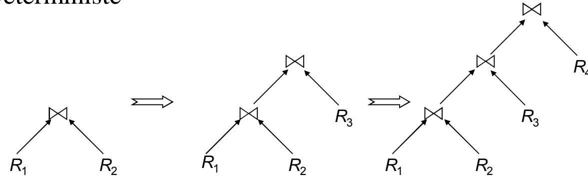


Stratégie de recherche

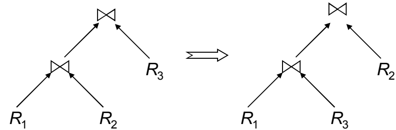
- Il est en général trop coûteux de faire une recherche exhaustive
- Déterministe
  - part des relations de base et construit les plans en ajoutant une relation à chaque étape
  - programmation dynamique: largeur-d'abord
  - excellent jusqu'à 5-6 relations
- Aléatoire
  - recherche l'optimalité autour d'un point de départ particulier
  - réduit le temps d'optimisation (au profit du temps d'exécution)
  - meilleur avec > 5-6 relations
    - **recuit simulé (simulated annealing)**
    - **amélioration itérative (iterative improvement)**

Stratégies de recherche

- Déterministe



- Aléatoire

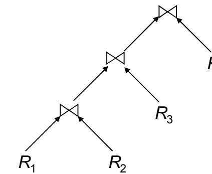


R1 R3 R2

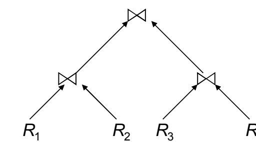
Algorithmes de recherche

- Limiter l'espace de recherche
  - heuristiques
    - par ex. appliquer les opérations unaires avant les autres
    - Ne marche pas toujours (perte d'index, d'ordre)
  - limiter la forme des arbres

Arbre linéaire



Arbre touffu



Génération de plan physique

- Sélection :
  - Commencer par les conditions d'égalité avec un index sur l'attribut
  - Filtrer sur cet ensemble de n-uplets ceux qui correspondent aux autres conditions
- Jointure
  - Utilisation des index, des relations déjà triées sur l'attribut de jointure, présence de plusieurs jointures sur le même attribut
- Pipelines ou matérialisation

Conclusion

- Point fondamental dans les SGBD
- Importance des métadonnées, des statistiques sur les relations et les index, du choix des structures d'accès.
- L'administrateur de bases de données peut améliorer les performances en créant de nouveaux index, en réglant certains paramètres de l'optimiseur de requêtes (voir TME et cours M1)