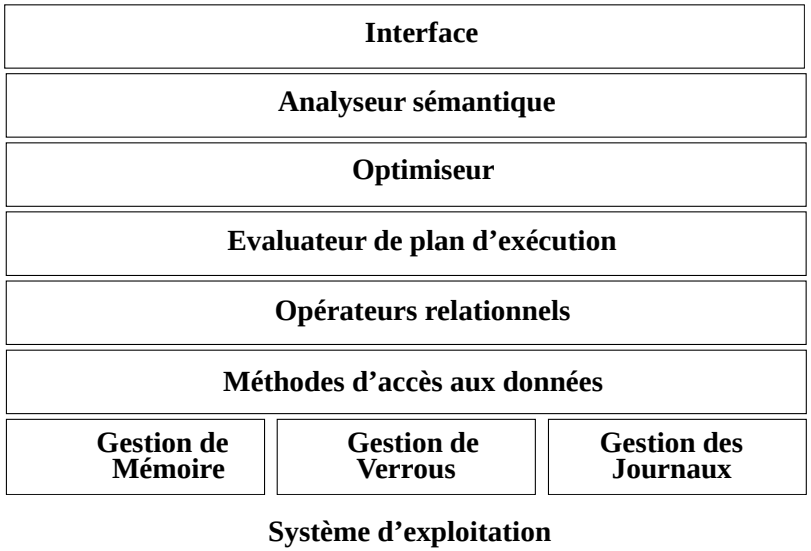
Optimisation de Requêtes

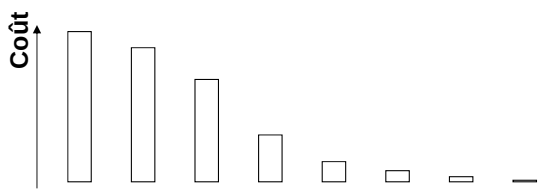
1. Introduction
2. Simplification de requêtes
3. Restructuration algébrique
4. Choix du meilleur plan
5. Conclusion

Architecture en couche d'un SGBD



Optimiser : pour quoi faire ?

- Une question
- Plusieurs expressions équivalentes en SQL
- Plusieurs expressions équivalentes en algèbre
- Plusieurs algorithmes équivalents

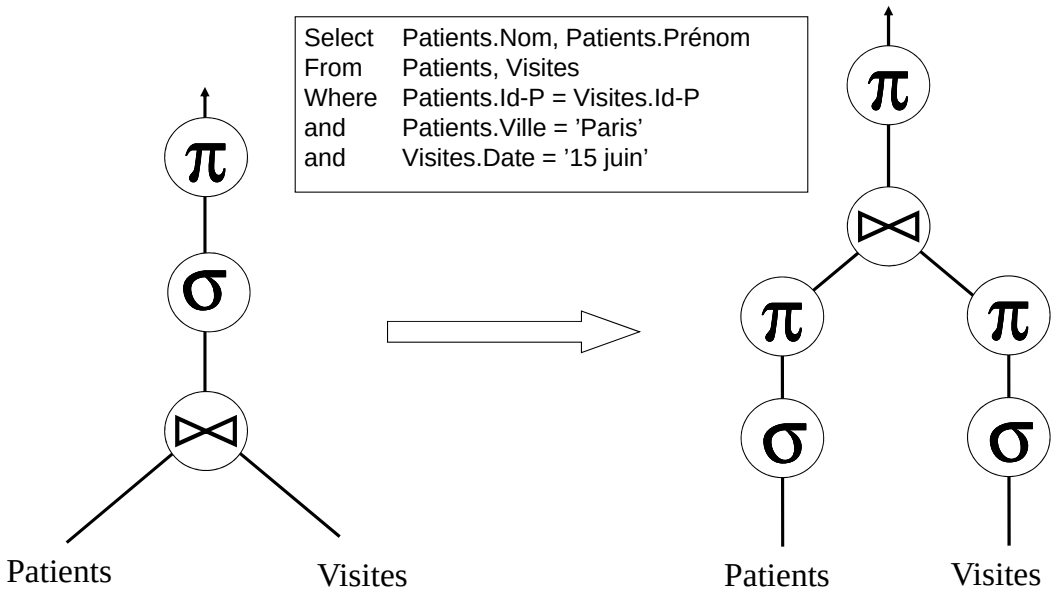


Plans sémantiquement équivalents
Variation de performance de plusieurs ordres de magnitude

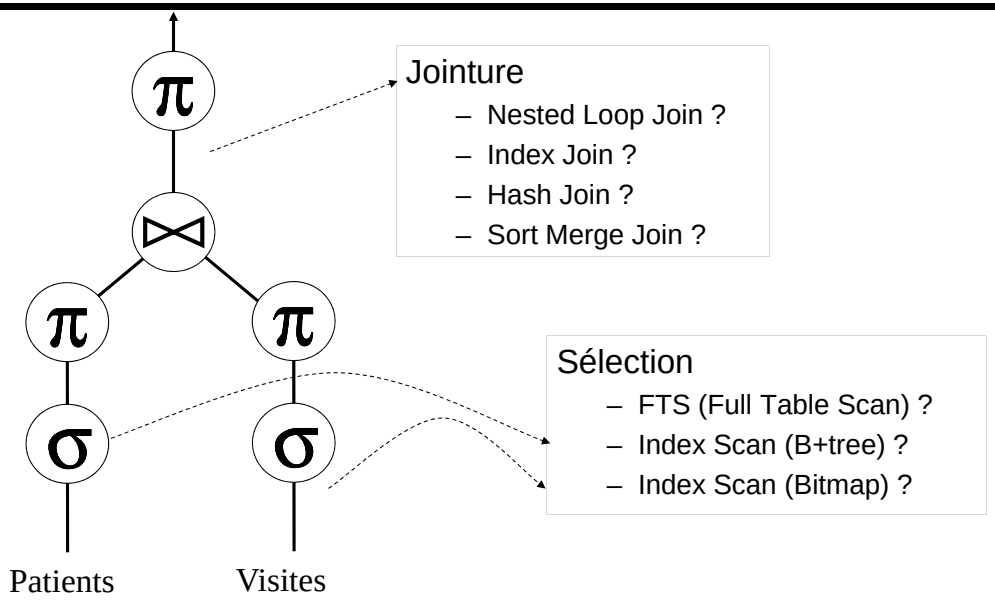
Ex:
5 tables ➔ 1620 plans possibles
10 tables ➔ 17 milliards de plans…

Optimisation : exemple

```
Select   Patients.Nom, Patients.Prénom
From     Patients, Visites
Where    Patients.Id-P = Visites.Id-P
and      Patients.Ville = 'Paris'
and      Visites.Date = '15 juin'
```



Optimisation : exemple



Jointure
- Nested Loop Join ?
- Index Join ?
- Hash Join ?
- Sort Merge Join ?

Sélection
- FTS (Full Table Scan) ?
- Index Scan (B+tree) ?
- Index Scan (Bitmap) ?

Qui optimise ?

- Dans la théorie …
  - 2 requêtes SQL équivalentes (i.e., donnant le même résultat) doivent, après optimisation, produire le même plan d'exécution (i.e., le meilleur) !
  - Seuls les concepteurs de SGBD (noyau) devraient avoir besoin de comprendre l'optimisation pour développer un bon optimiseur de requêtes
- Mais dans la pratique
  - Les algorithmes ont leurs limites (2 requêtes équivalentes peuvent ne pas être identifiées comme telles)
  - Le bon choix dépend souvent des données (distribution, taille, sélectivité), dont la connaissance est approximative
  - Le bon choix dépend également de ce que l'on souhaite optimiser (ressources, temps d'exécution, latence ?)
  - Enfin, seuls les plans d'exécution rendus possibles par le schéma physique de la BD sont évalués

➔le DBA joue un rôle majeur

Etape 1 : Simplification de requêtes (1)

1. Utilisation de la logique des prédicats

- Ex. Requête initiale

```
SELECT * FROM Medic M
WHERE ((M.label = 12)                          P
       OR (M.labo = 'WHITEHALL')               Q
       OR (M.labo = 'AVENTIS'))                R
       AND NOT ((M.labo = 'WHITEHALL')         Q
       OR (M.labo = 'AVENTIS ')) ;             R
```

- Critère: $((P \vee Q \vee R) \wedge \text{NOT } (Q \vee R)) \equiv P \wedge \text{NOT } Q \wedge \text{NOT } R$
- Requête équivalente avec critère simplifié

```
SELECT * FROM Medic M
WHERE (M.label = 12)                   P
AND NOT (M.labo = ' WHITEHALL')        Q
AND NOT (V.labo = 'AVENTIS');          R
```

Pourquoi une requête aussi stupide ?

1) c'est une requête sur une vue
2) une requête construite par programme
3) une requête générée par une IHM avec clics successifs ...

Simplification de requêtes (2)

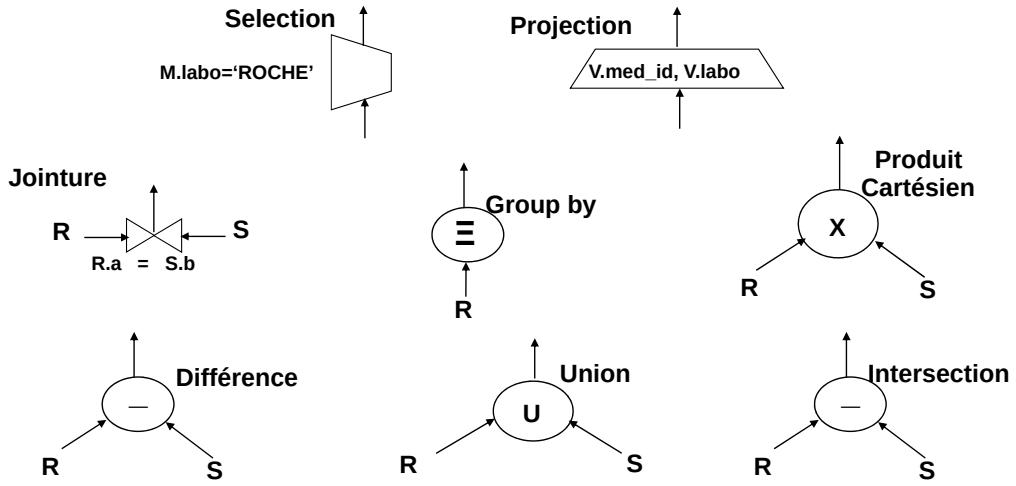
2. Utilisation de contraintes d'intégrité
   - Contraintes contradictoires avec la qualification
     - SELECT * FROM Medic WHERE labo = 'Roche' AND label < 10
     - Contrainte d'intégrité sur Medic : labo = 'Roche' ➔ label > 12
     - SELECT * FROM Medic WHERE labo = 'Roche'
       AND label < 10 AND label > 12

     ➔ Inutile d'exécuter la requête : 0 tuple résultat
   - Complément de la requête
     - SELECT * FROM Medic WHERE labo = 'Roche'
     - Il existe un index sur Medic.label (mais pas sur Medic.labo…)
     - SELECT * FROM Medic WHERE labo = 'Roche' AND label > 12

     ➔ L'index est utilisable pour évaluer la requête

Etape 2 : Restructuration algébrique

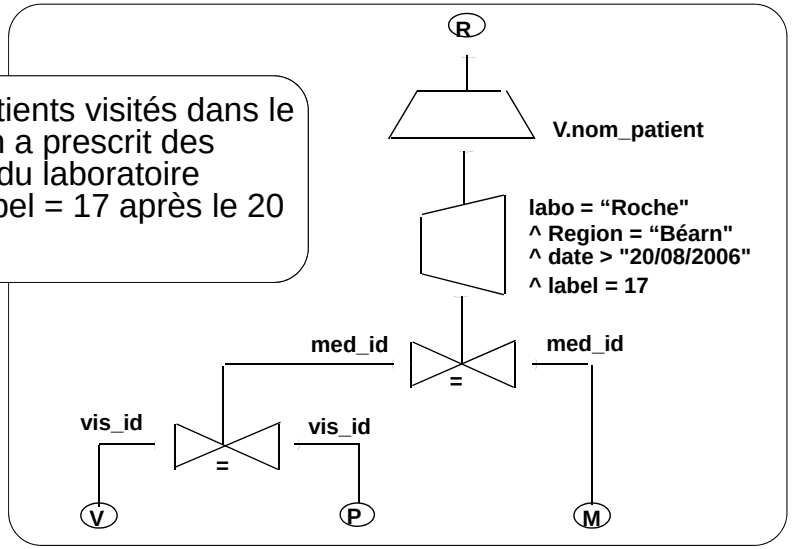
- Convention de représentation des opérateurs



Ex. Plan d'exécution candidat (1)

Requête

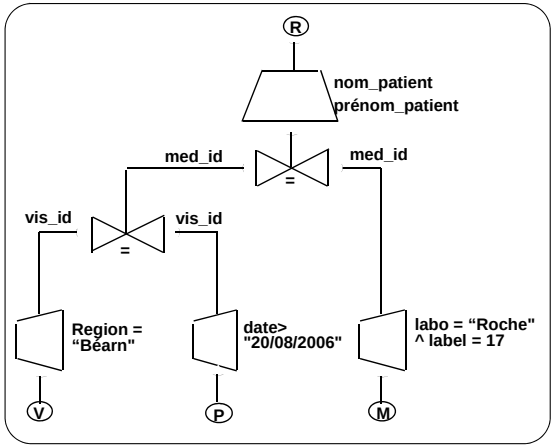
« Nom des patients visités dans le Béarn à qui on a prescrit des médicaments du laboratoire ROCHE de label = 17 après le 20 août 2006 »



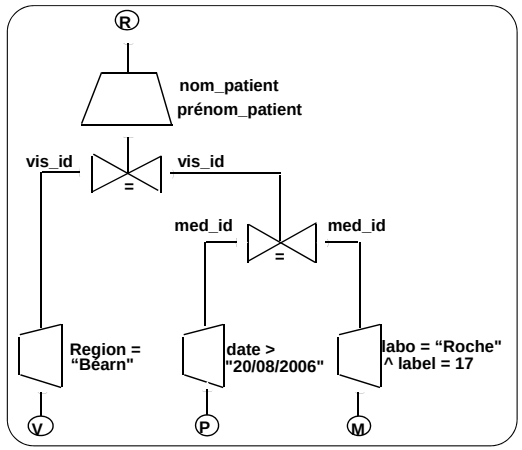
Plan candidat N°1

Ex. Plan d'exécution candidat (2)

Plan candidat N°2



Plan candidat N°3



De ces 3 arbres, lequel est le meilleur ?

Le premier est sûrement moins bon, mais les 2 derniers ?

Restructuration algébrique : objectif

- Problème
  - Suivant l'ordre des opérateurs algébriques dans un arbre, le coût d'exécution est différent
- Pourquoi ?
  - Certains opérateurs diminuent le volume de données alors que d'autres peuvent l'augmenter
  - Le coût des algorithmes varie en fonction du volume de données à traiter (progression logarithmique, linéaire, exponentielle)
  - Certains opérateurs/algorithmes changent l'organisation (tri, placement) des tables opérandes
- La restructuration algébrique
  - Consiste à transformer l'arbre pour en minimiser le coût
  - En exploitant les équivalences d'expression de l'algèbre relationnelle …

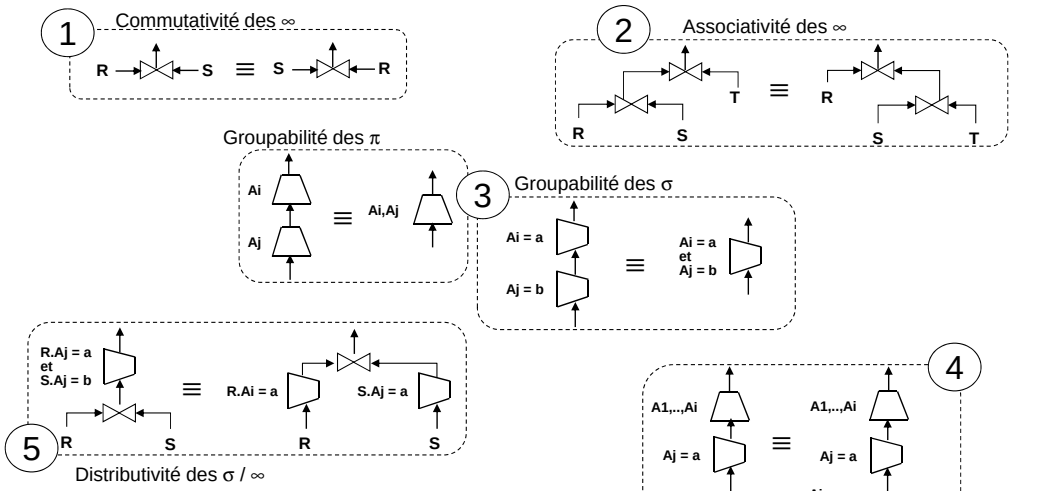
Equivalences de l'algèbre relationnelle

- Deux expressions algébriques sont équivalentes SSI elles produisent le même résultat
- Exemples de règles d'équivalence
  - Sélections $\sigma_{c1}(\sigma_{c2}(\dots \sigma_{cn}(Doc))) \equiv \sigma_{c1 \wedge c2 \wedge \dots \wedge cn}(Doc)$ (Groupement)
    $\sigma_{c1}(\sigma_{c2}(Doc)) \equiv \sigma_{c2}(\sigma_{c1}(Doc))$ (Commutativité)
  - Projections $\pi_{a1}(\pi_{a2}(\dots \pi_{an}(Doc))) \equiv \pi_{a1, \dots, an}(Doc)$ (Groupement)
  - Jointures $Doc \infty (Vis \infty Pres) \equiv (Doc \infty Vis) \infty Pres$ (Associativité)
    $Doc \infty Vis \equiv Vis \infty Doc$ (Commutativité)
    (etc…)
- Ces équivalence permettent notamment de
  - Changer l'ordre des opérations de jointure
  - 'Pousser' les sélection/projection avant les jointures

Autres règles d'équivalence

- Projection commute avec sélection si la projection conserve les attributs utilisés par la sélection (*Semi commutativité*)
  - Ex. $\pi_{Nom}(\sigma_{Nom>'\%D'}(Doc)) \equiv \sigma_{Nom>'\%D'}(\pi_{Nom}(Doc))$
- Une sélection sur R commute avec R∞S (*distributivité de σ/∞*)
  - $\sigma_{C1}(R \infty S) \equiv (\sigma_{C1}(R)) \infty S$, avec C1 critère sur attributs de R
  - $\sigma_{C1}(R \infty S) \equiv (\sigma_{C1}(R)) \infty (\sigma_{C1}(S))$, avec C1 critère sur attributs de R et S
- Si une projection suit une jointure R∞S, on peut la 'pousser' en retenant seulement les attributs de R (et de S) nécessaires à la jointure (ou aux projection finales) (*semi-distributivité de π/∞*)
  - Ex. $\pi_{Nom, Date}(Doc \infty Vis) \equiv \pi_{Nom}(\pi_{Nom, id}(Doc) \infty \pi_{Date, docid}(Vis))$

Bilan sur les règles d'équivalences



- Et aussi…
  - Semi-distributivité des π / ∞ (6)
  - Distributivité des σ / Unions & différences (7)
  - Distributivité des π / Unions (8)

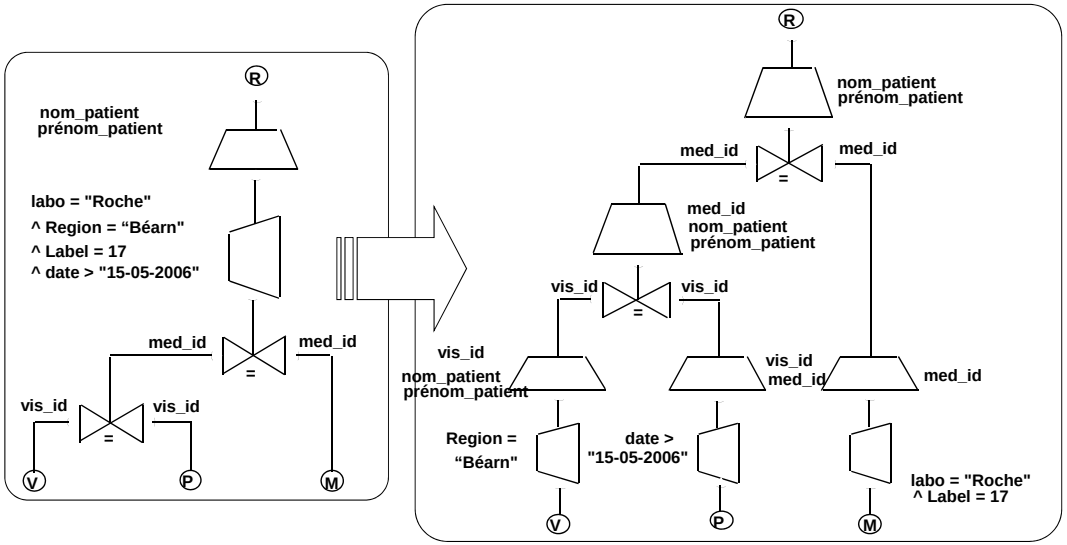
Heuristiques d'optimisation

- Appliquer d'abord les opérations réductrices (sélections et projections) en les groupant par table
  - Dit autrement : « secouer l'arbre » pour faire tomber les opérations « lourdes » près de la racine

  1. Dégrouper les sélections (Règle 3)
  2. Rapprocher les sélections des feuilles (Règles 4, 5 et 7)
  3. Grouper les sélections aux feuilles (Règle 3)
  4. Rapprocher les projections des feuilles (Règles 4, 6 et 8)

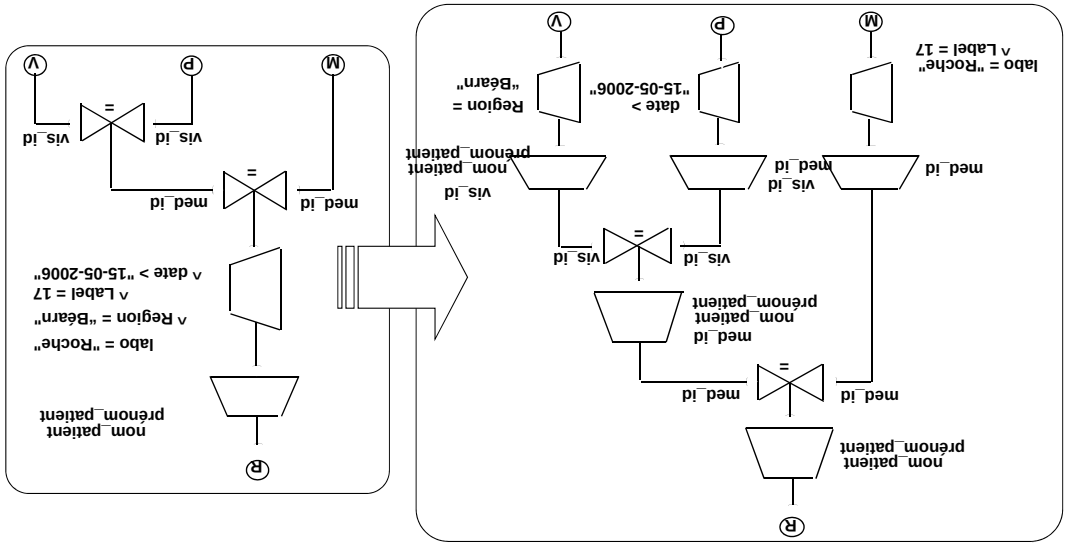
- L'ordre des unions, différences et jointures est pour l'instant inchangé !!!

Exemple d'optimisation



Est-ce une bonne opération d’ajouter des projections partout ?

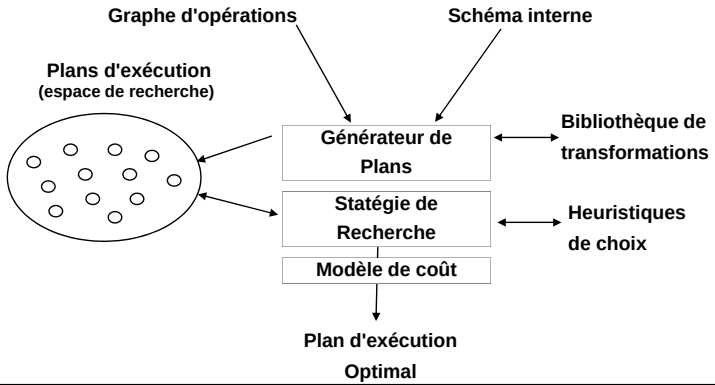
Exemple d'optimisation



Est-ce une bonne opération d’ajouter des projections partout ?

Optimisation basée sur les coûts

- Rule-Based Optimizer (ex: RBO d’Oracle)
  - Basé exclusivement sur des heuristiques
- Cost-Based Optimizer (ex: CBO d’Oracle)
  - modèle de coût + statistiques sur les données
  - Plus précis mais plus complexe



Difficultés de l’optimisation basée coût

- Espace de recherche très grand (plans candidats)
  - n algorithmes par opérateur
  - p ordonnancement pour les opérations binaires
    - Sans considérer les algorithmes, il y a 1620 ordres possibles pour joindre 5 tables, et 17 milliards pour 10 tables !
- Modèle de coût (choix du plan)
  - Difficulté pour estimer le coût de chaque opérateur
  - Difficulté encore plus importantes pour estimer la taille des résultats intermédiaires (permettant de calculer l’opérateur suivant)
  - Propagation exponentielle des erreurs (dans l’arbre d’exécution) !

Estimation des sélectivités

- TAILLE (σ(R)) = s * TAILLE(R) avec:
  - s (A = valeur) = 1 / NDIST(A) // NDIST = nombre de valeurs distinctes
  - s(A > valeur) = (max(A) - valeur) / (max(A) - min(A))
  - s (A IN liste valeurs) = (1/NDIST(A)) * CARD(liste valeurs)
  - s(P et Q) = s(P) * s(Q)
  - s(P ou Q) = s(P) + s(Q) - s(P) * s(Q)
  - s(not P) = 1 - s(P)
- TAILLE(R |><| S) = p * TAILLE(R) * TAILLE(S)
  - p = 1 / MAX(NDIST(A),NDIST(B)) si distribution uniforme des attributs A et B sur un même domaine
  - p = 1 si produit cartésien
- Modèle de coût basé sur des hypothèses d'uniformité de distribution et/ou sur des statistiques
  - RunStat(<Table>, <attribut>) : construction et stockage d'un histogramme

Conclusion

- L'optimisation de requêtes est une tâche cruciale du SGBD
  - Fort impact sur les performances du système
  - Mécanisme puissant mais complexe à maîtriser
- Le SGBD ne peut pas tout
  - Sans bon DBA, pas de bonne optimisation possible
- De plus en plus d'aide à l'administration
  - Outils d'audit des performances, 'explainer' de plans d'exécution, outils de recommandation de schéma physique (fragmentation de tables, index, vues concrètes, statistiques …)
    - Automatic SQL Tuning dans Oracle 10
    - Database Tuning Advisor dans SQL Server 2005
  - Longue route vers un futur SGBD auto-administrable