Bases de données avancées
Optimisation des opérateurs

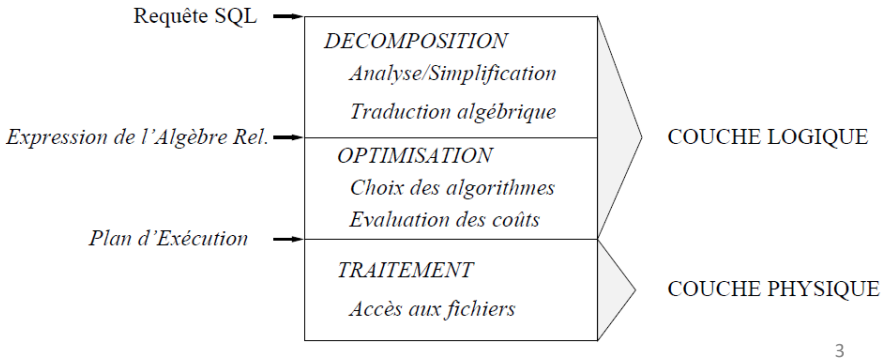
Sarah Cohen-Boulakia
Laboratoire de Recherche en Informatique
Université Paris Sud
http://www.lri.fr/~cohen/BD.html

Traitement d'une requête

- Requêtes exprimées en SQL: *langage déclaratif*
  - On indique *ce que l'on veut* obtenir
  - On ne dit pas *comment* l'obtenir
- Le SGBD doit faire le reste
  - Déterminer la façon d'exécuter la requête: *plan d'exécution*
  - Plusieurs plans possibles → choisir le meilleur possible : *optimisation*
  - Exécuter le plan choisi : *évaluation*
- Plan d'execution
  - Exprimé en *algèbre* relationnelle (expression algébrique)
  - Forme *exécutable*: on sait précisément *comment* l'évaluer

Les étapes

- **Décomposition**: *requête SQL exprimée en algèbre relationnelle*
- **Optimisation**: *expression algèbre relationnelle → plan d'exécution*
- **Évaluation** (traitement): *plan d'exécution résultats*



Décomposition (1/2)

- 5 Sous-étapes
  - Analyse syntaxique, Analyse sémantique
  - Simplification, Normalisation (forme canonique)
  - Traduction algébrique
- Analyse syntaxique
  - Requête = chaîne de caractères
  - Contrôle de la structure grammaticale (syntaxe SQL)
  - Vérification de l'existence des rel/att adressés dans la requête
    - Utilisation du dictionnaire de données de la base
  - Transformation en une représentation interne : arbre syntaxique

Décomposition (2/2)

- Analyse sémantique
  - Vérification des opérations réalisées sur les attributs
    Ex. Pas d'addition sur un attribut texte
  - Détection d'incohérences
    - *Ex. prix*=5 *and prix*=6
- Simplification
  - Conditions inutilement complexes
    Ex. (A or not B) and B est équivalent à (*A and B)*
  - Peut mettre en jeu les contraintes d'intégrité (pas toujours)
    Ex. prix < 300 AND age < 18, si on sait que age < 18 → prix < 150
    on remplace par prix < 150 AND age < 18
- Normalisation
  - Simplifie la traduction algébrique
    - Transformation des conditions en forme normale conjonctive
      - (A OR ….) AND (D OR …) AND …
    - Décomposition en blocs *Select-From-Where*

Traduction algébrique

- Déterminer une expression algébrique équivalente à la requête SQL
- Clause SELECT opérateur de projection
- Clause FROM les relations qui apparaissent dans l'expression
- Clause WHERE
  - Condition "*Attr = constante*" opérateur de sélection
  - Condition "*Attr1 = Attr2*" jointure ou sélection
- Résultat: expression algébrique
  - Représentée par un *arbre de requête*

Exemple de traduction algébrique

- Soit le schéma relationnel (simplifié)
  - ***Cinéma*** (*ID-cinéma*, nom, adresse)
  - ***Salle*** (*ID-salle*, ID-cinéma, capacité)
  - ***Séance*** (*ID-salle*, *heure-début*, film)
- Requête: **quels films commencent au Multiplex à 20 h ?**

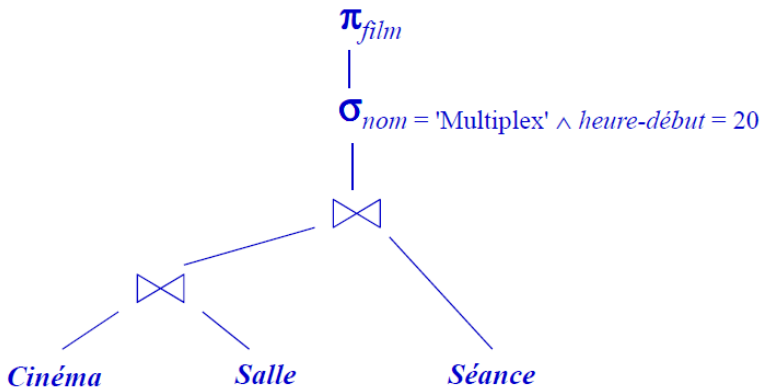
```
SELECT Séance.film
FROM Cinéma, Salle, Séance
WHERE Cinéma.nom = 'Multiplex'
AND Séance.heure-début = 20
AND Cinéma.ID-cinéma = Salle.ID-cinéma
AND Salle.ID-salle = Séance.ID-salle
```

- **Expression algébrique**

$$\pi_{film}\,(\sigma_{nom\,=\,\text{'Multiplex'}\,\wedge\,heure\text{-}début\,=\,20}\,((Cinéma \bowtie Salle) \bowtie Séance)$$

Arbre de requête

$$\pi_{film}\,(\sigma_{nom\,=\,\text{'Multiplex'}\,\wedge\,heure\text{-}début\,=\,20}\,((Cinéma \bowtie Salle) \bowtie Séance)$$



Optimisation

- Pour une requête SQL, il y a plusieurs expressions algébriques *équivalentes* possibles
- Le rôle de principe de l'optimiseur
  - Trouver les expressions équivalentes à une requête
  - Évaluer leurs coûts et choisir « la meilleure » (ou plutôt une qui n'est pas la pire ☺)
- On passe d'une expression à une autre équivalente en utilisant des **règles de réécriture**

Equivalences algébriques

Commutativité et associativité de la jointure

$$E_1 \bowtie E_2 = E_2 \bowtie E_1,$$
$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3).$$

Cascade de projections

$$\pi_{A_1,\ldots,A_n}(\pi_{B_1,\ldots,B_m}(E)) = \pi_{A_1,\ldots,A_n}(E)$$

Cascade de sélections

$$\sigma_{F_1}(\sigma_{F_2}(E)) = \sigma_{F_1 \wedge F_2}(E)$$

Commutation sélection et projection
Si F ne porte que sur $A_1, \ldots, A_n$,

$$\pi_{A_1,\ldots,A_n}(\sigma_F(E)) = \sigma_F(\pi_{A_1,\ldots,A_n}(E))$$

Si F porte aussi sur $B_1, \ldots, B_m$,

$$\pi_{A_1,\ldots,A_n}(\sigma_F(E)) = \pi_{A_1,\ldots,A_n}(\sigma_F(\pi_{A_1,\ldots,A_n,B_1,\ldots,B_m}(E)))$$

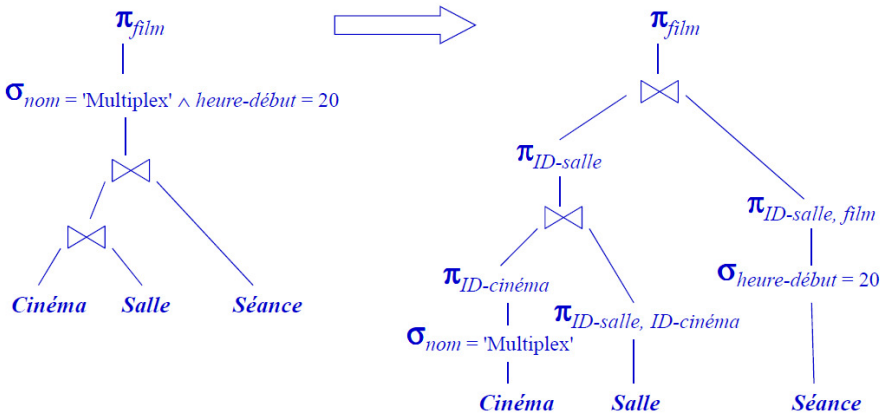
Réécriture algébrique

- Transformation de l'expression algébrique pour en obtenir une autre "meilleure" équivalente
  - Utilisation des règles de réécriture
- Exemple d'algorithme de restructuration
  1. Séparer les sélections à plusieurs prédicats en plusieurs sélections à un prédicat (*de*groupement des sélections)
  2. Descendre les sélections le plus bas possible dans l'arbre (règles de commutativité et distributivité de la sélection)
  3. Regrouper les sélections sur une même relation (regroupement des sélections)
  4. Descendre les projections le plus bas possible dans l'arbre (règles de commutativité et distributivité de la projection)
  5. Regrouper les projections sur une même relation

Justification de l'algorithme

- Idée de base: **réduire la taille des données** traitées le plus tôt possible
  - On réalise d'abord les sélections, car c'est le plus souvent l'opérateur le plus « réducteur »
  - On réalise dès que possible des projections pour éliminer les attributs inutiles
  - On réalise les jointures (opération très coûteuse) une fois que la taille des données a été réduite au maximum
- Question: le plan ainsi obtenu est-il toujours optimal?
- Réponse: NON, d'autres facteurs peuvent intervenir

Exemple de restructuration



Conclusions réécriture algébrique

- La réécriture algébrique est nécessaire, mais pas suffisante
- Il faut tenir compte d'autres critères
  - *Les chemins d'accès* aux données
    - On peut accéder aux données d'une table par accès séquentiel (*full scan)*, par index, etc.
  - *Les différents algorithmes* possibles pour réaliser un opérateur
    - Par exemple, plusieurs algorithmes pour la jointure
    - Ces algorithmes dépendent des chemins d'accès disponibles
  - *Les propriétés statistiques* de la base de données
    - Taille des tables
    - Sélectivité des attributs
    - etc.

Chemins d'accès à une table

- Dépendent de l'organisation physique de la table
  - *Accès séquentiel*: toujours possible (*full scan*)
    - Coût = nb pages de la table (aussi appelé nb d'E/S)
  - *Accès par index*
    - Pour chaque index sur un attribut *A* de la table
      - Valeur *v* de *A* → liste d'adresses (ROWID) des articles ayant *A=v*
      - Intervalle de valeurs [*v1,v2*] de *A* → liste d'adresses des articles ayant *A* dans [*v1,v2*]
    - Pour chaque index sur une liste d'attributs (*A1*, *A2*, …, *An*)
      - Valeurs *vi* de *Ai* (1 <= *i* <= *n*) → liste d'adresses des articles ayant *Ai=vi*
      - Remarque: un index sur (***A1***, *A2*, …, *An*) est utilisable aussi comme index sur (***A1***, *A2*, …, *Ak*), *k < n mais on ne peut pas l'utiliser pour Ai si Ai-1 n'a pas de valeur (Plus en TD)*

- Coût = coût d'accès à l'index + coût d'accès au disque (si necessaire)

Algorithmes de jointure

- La jointure est l'opération la plus coûteuse
  - Son optimisation est très importante
- Plusieurs algorithmes, dépendants du chemin d'accès
  - Chacun peut être meilleur dans des situations spécifiques
  - Choix entre plusieurs algorithmes pour tendre vers la meilleure optimisation
- Dans la suite
  - R et S sont deux tables
  - R : M pages, pR enregistrements/page
  - S : N pages, pS enregistrements/page
  - Rappel : enregistrement = ligne = tuple = n-uplet
- On dispose toujours de 3 emplacements : une page pour R, une pour S et une pour la sortie

Jointure itérative brute

- **Idée :** Evaluation tuple par tuple

  Pour tout tuple r dans R faire

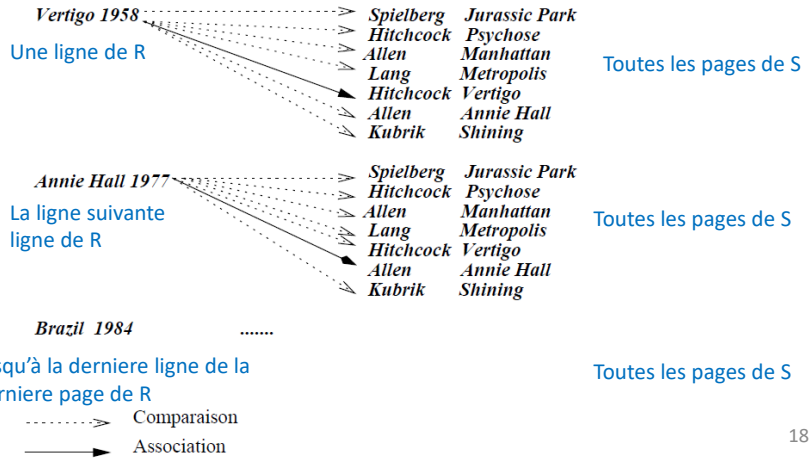
  Pour tout tuple s dans S faire

  si r.a==s.a alors ajouter <r,s> au résultat

- On parcourt la relation R autant de fois que R a de pages : M E/S
- On parcourt S autant de fois que R a de tuples (pR*M) et à chaque passage on est obligé de charger toutes les pages de S (N)
- **Au total :** M + pR*M*N

Exemple de jointure par boucles imbriquées

- ***Film*** (titre, année) ▷◁ ***Réalisateur*** (nom, titre)



Jointure page à page

- **Idée :** On exploite la structure en pages de R et S. Pour chaque page de R on peut retrouver chaque page de S et écrire les tuples <r,s> satisfaisant la condition de jointure.
- Pour chaque page de R faire

  Pour chaque page de S faire

  si r.a==s.a alors ajouter <r,s> au résultat
- On gagne un facteur de pR
- **Au total :** M + M*N
- **Optimisation :** on choisit la relation la plus petite en nb de pages (telle que M<N)

Jointure par bloc

- **Idée :** décomposer la plus petite relation en blocs stockés dans les pages du buffer et construire en mémoire une structure pour les blocs de R.
- Pour tout bloc de B pages de R faire

  Pour toute page de S faire

  si r.a==s.a alors ajouter <r,s> au résultat

  (r dans R-bloc et s dans S-page)

- Coût du parcours de R : M
- Le parcours de S se fait M/B fois (partie entière sup). Et chaque parcours coûte toujours N.
- **Au total :** M +N* (M/B)
- Il existe une technique de double buffering

Jointure avec index

- **Idée** : Pour chaque tuple de R on utilise l'index pour retrouver les tuples *correspondant* de S (c'est S qui doit être indéxée). On compare r seulement avec les tuples de S qui ont la même valeur sur la colonne de jointure

  → On n'énumère donc pas le produit cartésien.

- Pour chaque tuple de R faire

  Pour chaque tuple de s où r.a==s.a

  faire ajouter <r,s> au résultat

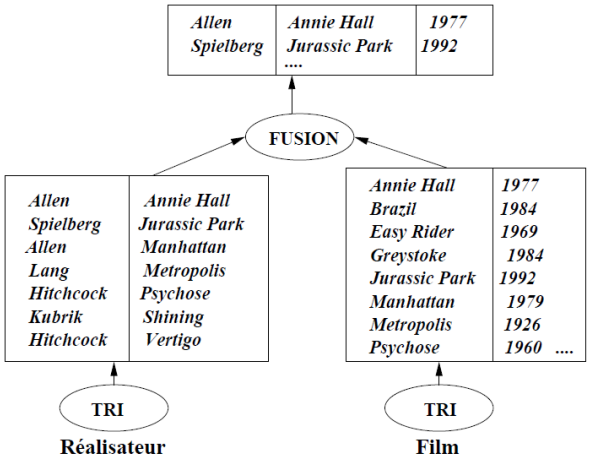
- Le coût pour scanner R est toujours M
- Le coût pour retrouver les tuples correspondant dans S dépend du type d'index et du nombre de tuples « correspondant » (*matching tuples*).
- **M + PR * M * (coût d'accès index + coût index ↦ données)**

Jointure par tri

- **Idée :** On trie les 2 relations sur l'attribut de jointure puis on recherche les tuples satisfaisant la condition de jointure en fusionnant les 2 relations
  - On regroupe tous les tuples avec la même valeur de colonne de jointure
  - On exploite ce « partitionnement » en comparant les R tuples d'une partition avec seulement les S tuples de la même partition
- Coût = coût du tri (et pas toujours besoin) + coût de la jointure
- **Coût** =(MlogM + NlogN) + M + N

Exemple de jointure par tri-fusion

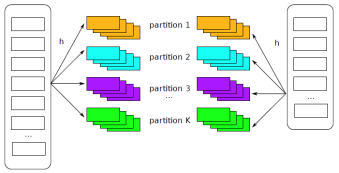
Réalisateur (nom, titre) ▷◁ ***Film*** (titre, année)



Jointure par hachage

- On choisit une fonction de hachage h et on partitionne R selon h(r.a) pour obtenir K partitions
- On partitionne S selon h(s.a) pour obtenir K partitions
- Idée : Si deux tuples r.a et s.a doivent être joints alors
  - on a $h(r.a) = h(s.b) = u$
  - et on trouvera ces tuples dans la partition u de R et la partition u de S.

→Il suffit d'effectuer la jointure sur les paires de fragments correspondant à **la même valeur de la fonction de hachage**.



Jointures généralisées

- **Égalité sur plusieurs attributs** (R.a = S.a AND R.b = S.b)
  - Jointure itérative par index: on peut choisir de créer un index pour S sur (a,b) ou utiliser des indexes existants sur l'un ou l'autre
  - On peut aussi utiliser jointure par tri-fusion et hachage en utilisant (a,b) comme clé de tri/hachage
- Conditions d'**inégalité**
  - Pour les jointures par index, il faut un arbre B+ groupant (sinon sur-coût pour aller chercher les données)
  - Jointure par tri-fusion et hachage impossible
  - Jointure itérative par bloc est la meilleure option en général

Selectivité

- Taux (ou facteur) de sélectivité d'une condition ϕ (ou d'une requête) pour une relation donnée:

$$\frac{\text{\# d'enregistrements sélectionnés}}{\text{\# d'enregistrements}}$$

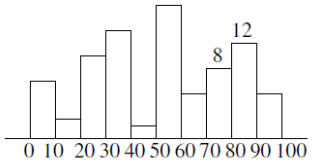
- Le choix de certains algorithmes dépend de la sélectivité
- On ne connaît la « vraie » valeur de la sélectivité qu'après avoir évalué la requête
- On utilise des statistiques sur les relations pour tenter une approximation du taux de sélectivité

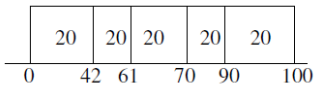
Statistiques sur les relations

- Le SGBD conserve, entre autres, les statistiques suivantes pour chaque relation
  - Nombre d'enregistrements (N), taille d'un enregistrement, nombre d'attributs/page (P)
  - Nombre de pages de la relation (les pages ne sont pas toutes remplies de manière optimale)
  - V(a) : nombre de valeurs distinctes pour l'attribut a (dans la relation R)
  - Estimation de sélectivité pour l'attribut a: V(a)/N
  - Profondeur pour les arbres B+
  - Nombre de pages pour les feuilles d'un arbre B+
  - Nombre de valeurs distinctes pour la clé de recherche d'un index…

Sélectivité et statistiques

Histogramme en hauteur
Les valeurs prises par un attribut sont triées puis divisées en n intervalles égaux : chaque intervalle → nb d'enregistrements pour lesquels l'attribut a une valeur dans l'intervalle





Histogramme en largeur : On trie les enregistrements on définit les n intervalles pour que chacun contienne le même nb d'enregistrements

Sélectivité et index

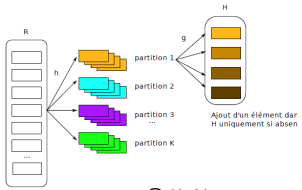
- Sélectivité : nb d'enregistrements pertinents relativement aux critères de recherche / nb total d'enregistrements
- Attention : les SGBD n'exploitent la séléctivité qu'en cas d'utilisation d'index !
- Si un index porte sur un attribut alors
  - l'évaluation du cout de la sélection se fait sur la sélectivité de l'attribut
- Si un index porte sur plusieurs attributs (A,B,…) alors
  - La sélectivité n'agit que s'il y a au moins une condition sur A
  - Si inégalité sur A et B, une seule sélectivité agit (celle de A)
  - Si égalité sur A et condition sur B (inégalité ou égalité) alors la double sélectivité s'applique (A * B)

Projection

- Projection : *à la volée*
- Projection (π) de l'algèbre relationnelle **sans doublon**
- SELECT DISTINCT a,b FROM t
  - Si index sur (a,b) disponible, utilisation directe de l'index
  - Sinon tri et projection durant la phase de tri
  - Double partitionnement par hachage

Double partitionnement

- Deux fonctions de hachage h et g distinctes
- On partitionne R en K partitions en utilisant h
- Pour chaque partition entre 1 et K
  - On crée une table de hachage en mémoire (avec g comme fonction) pour éliminer les doublons de la partition
- On supprime les doublons sans trier



© K. Nguyen

Autres opérateurs relationnels

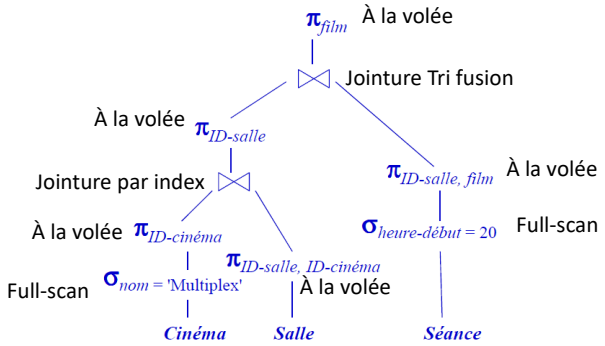
- Intersection et produit cartésien: cas *dégénérés* de jointure
- UNION DISTINCT et EXCEPT sont similaires.
  - Approche par tri
    - On trie les deux relations sur tous les attributs
    - On fusionne en éliminant les doublons.
    - Résultat trié
  - Approche par hachage
    - Double partitionnement.
    - On partitionne R et S avec h.
    - Pour chaque partition de S et R, on ajoute les éléments dans une table H, en éliminant les doublons

Et pour finir… les agregats

- Sans clause GROUP BY
  - scan de la relation
  - Si les attributs agrégés sont dans un index, on peut faire un scan d'index uniquement (en espérant que l'index soit plus petit!)
- Avec GROUP BY
  - identique au cas sans GROUP BY mais tri préalable pour déterminer les groupes, et scan « groupe par groupe » pour calculer la fonction d'agrégat

Un arbre complet

- On indique les algorithmes utilisés pour chaque opérateur



En plus

Exemple d'optimisation qui échoue

- On considère une table *Film*, et une table *Séance*
  - ***Film*** (*film*, réalisateur, année)
- Requête: les réalisateurs des films qu'on peut voir entre 14h-22h

```
SELECT Film.réalisateur
FROM Film, Séance
WHERE Séance.heure-début > 14
AND Séance.heure-début < 22
AND Film.film = Séance.film
```

- Expressions algébriques
  - Initiale: $\pi_{réalisateur}\ (\sigma_{\ heure\text{-}début\ >\ 14\ \wedge\ heure\text{-}début\ <\ 22}\ (\boldsymbol{Film} \rhd\lhd \boldsymbol{Séance}))$
  - Optimisée: $\pi_{réalisateur}\ (\boldsymbol{Film} \rhd\lhd \sigma_{\ heure\text{-}début\ >\ 14\ \wedge\ heure\text{-}début\ <\ 22}\ (\boldsymbol{Séance}))$
- Hypothèses

– *Film* occupe 8 pages et ne contient que pour 20% des films de *Séance*
– *Séance* occupe 50 pages et 90% des séances sont entre 14h et 22h

Suite de l'exemple

Plan Initial $\pi_{réalisateur}(\sigma_{heure\text{-}début > 14 \wedge heure\text{-}début < 22}(Film \rhd\lhd Séance))$

– Jointure (brute): on lit 8 * 50 = 400 pages et on produit 20% * 50 = 10 pages

– Sélection: on produit 90% * 10 = 9 pages de séances entre 14h-22h

– On laisse de côté la projection (même coût dans les deux cas)

Coût (E/S): 400E + 10S + 10E + 9S = 429 E/S

Plan optimisé $\pi_{réalisateur}(Film \rhd\lhd \sigma_{heure\text{-}début > 14 \wedge heure\text{-}début < 22}(Séance))$

– Sélection: on lit 50 pages et on produit 90% * 50 = 45 pages de séances

– Jointure: on lit 8 * 45 = 360 pages et on produit 20% * 45 = 9 pages

Coût (E/S): 50E + 45S + 360E + 9S = 464 E/S

Le plan initial est ici meilleur que celui optimisé!

Cas rare: ici la jointure est plus sélective que la sélection!