Bases de données avancées

Évaluation et optimisation des requêtes

Dan VODISLAV

CY Cergy Paris Université

Master Informatique M1

Cours BDA

Plan

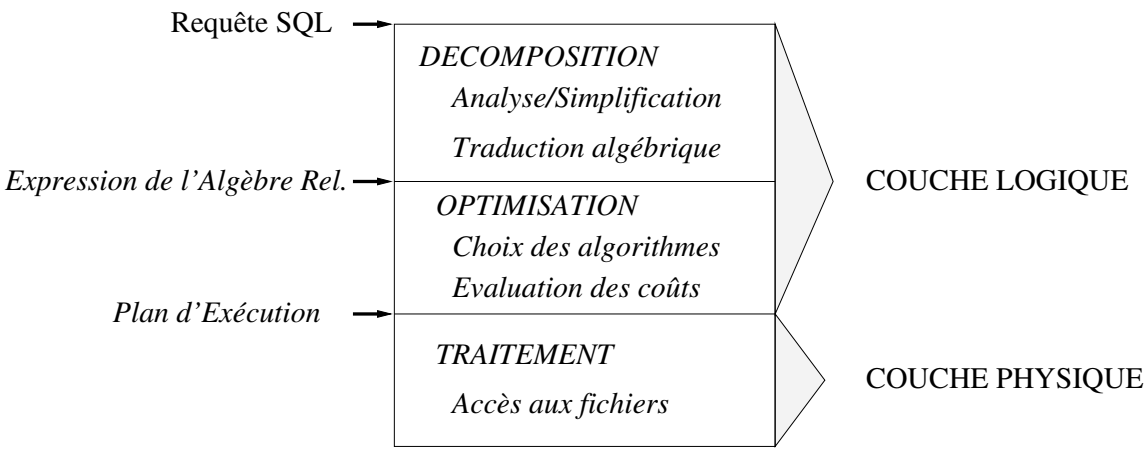
- Etapes de traitement d'une requête
  - Décomposition
  - Traduction algébrique et plan d'exécution
- Optimisation des requêtes
  - Principes
  - Règles de réécriture
  - Chemins d'accès
  - Algorithmes de jointure: boucles imbriquées, tri fusion, hachage, index
  - Algèbre physique
- Évaluation des requêtes
- Optimisation dans Oracle

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 : *optimisation*
  - Exécuter le plan choisi: *évaluation*
- Plan d'exécution
  - Exprimé en *algèbre* relationnelle (expression algébrique)
  - Forme *exécutable*: on sait précisément *comment* l'évaluer

Étapes

- **Décomposition**: *requête SQL → expr. algèbre relationnelle*
- **Optimisation**: *expr. algèbre relationnelle → plan d'exécution*
- **Évaluation** (traitement): *plan d'exécution → résultats*



Décomposition

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

Décomposition (suite)

- Analyse sémantique
  - Vérification des opérations réalisés 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*
- Normalisation: simplifie la traduction algébrique
  - Transformation des conditions en forme normale conjonctive
  - Décomposition en blocs *Select-From-Where*

Traduction algébrique

- Produire 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* = plan d'exécution de l'expression algébrique relationnelle
  - Point d'entrée dans la phase d'optimisation

Exemple de traduction algébrique

- Soit le schéma relationnel (notation simplifiée) :

  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 heures?

```
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
```

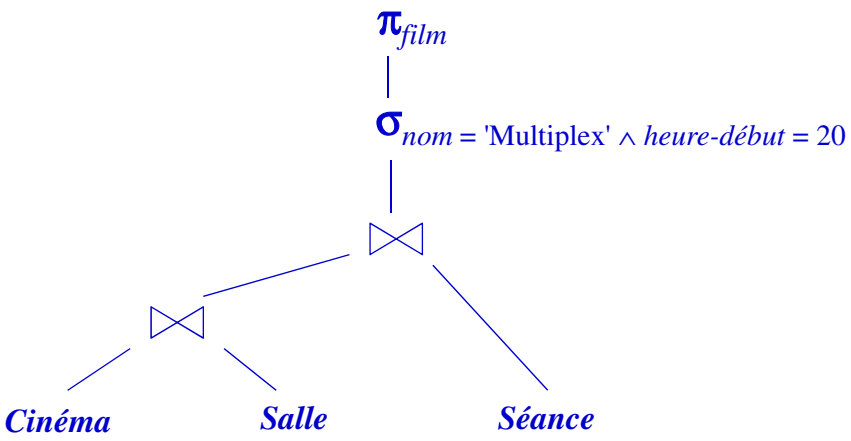
- Expression algébrique

  $\pi_{film}(\sigma_{nom = \text{'Multiplex'} \wedge heure\text{-}début = 20}((\textbf{\textit{Cinéma}} \rhd\lhd \textbf{\textit{Salle}}) \rhd\lhd \textbf{\textit{Séance}})$

Exemple (suite)

- Arbre de requête

$\pi_{film}\ (\sigma_{nom = \text{'Multiplex'} \wedge heure\text{-}début = 20}\ ((\textbf{\textit{Cinéma}} \rhd\lhd \textbf{\textit{Salle}}) \rhd\lhd \textbf{\textit{Séance}})$



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"
- On passe d'une expression à une autre équivalente en utilisant des **règles de réécriture**

Règles de réécriture

- De nombreuses règles existent
- Exemples
  - Commutativité des jointures

    $R \bowtie S \equiv S \bowtie R$
  - Associativité des jointures

    $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
  - Regroupement des sélections

    $\sigma_{A='a' \wedge B='b'}(R) \equiv \sigma_{A='a'}(\sigma_{B='b'}(R))$
  - Commutativité de la sélection et de la projection

    $\pi_{A1,\ldots,An}(\sigma_{Ai='a'}(R)) \equiv \sigma_{Ai='a'}(\pi_{A1,\ldots,An}(R))$, $i \in \{1, \ldots, n\}$

Règles de réécriture (suite)

- Exemples (suite)
  - Commutativité de la sélection et de la jointure

    $\sigma_{A='a'}(R(\ldots, A, \ldots) \bowtie S) \equiv \sigma_{A='a'}(R) \bowtie S$
  - Distributivité de la sélection sur l'union (pareil pour la différence)

    $\sigma_{A='a'}(R \cup S) \equiv \sigma_{A='a'}(R) \cup \sigma_{A='a'}(S)$
  - Commutativité de la projection et de la jointure

    $\pi_{A1,\ldots,An,B1,\ldots,Bm}(R \bowtie_{Ai=Bj} S) \equiv \pi_{A1,\ldots,An}(R) \bowtie_{Ai=Bj} \pi_{B1,\ldots,Bm}(S)$
  - Distributivité de la projection sur l'union (pareil pour la différence)

    $\pi_{A1,\ldots,An}(R \cup S) \equiv \pi_{A1,\ldots,An}(R) \cup \pi_{A1,\ldots,An}(S)$

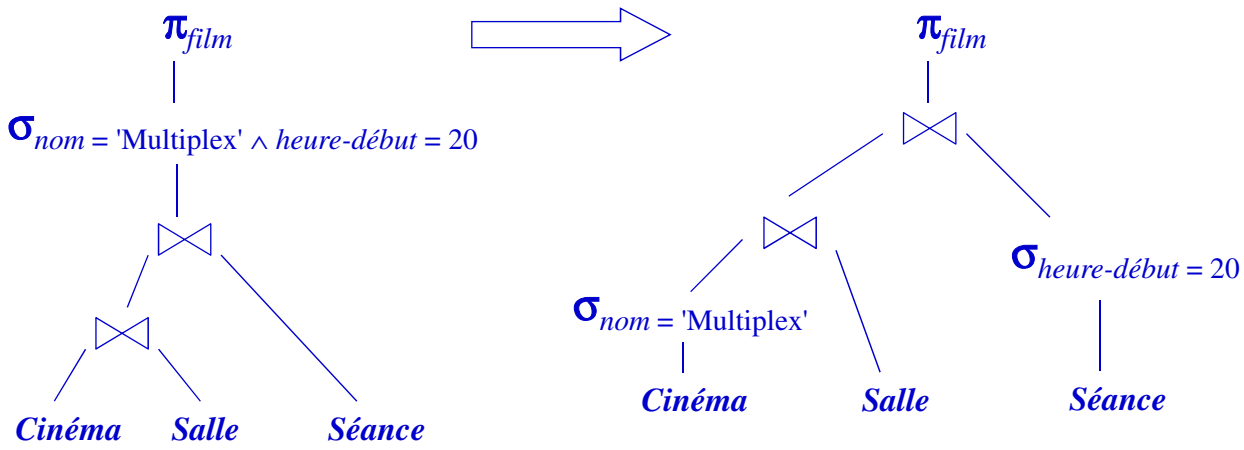
Restructuration (réécriture) algébrique

- Transformation de l'expression algébrique pour en obtenir une autre "meilleure" équivalente
  - Utilisation des règles de réécriture
- Quelles transformations choisir? → choix heuristiques
- Exemple d'algorithme de restructuration simple
  1. Séparer les sélections à plusieurs prédicats en plusieurs sélections à un prédicat (règle de regroupement 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 (règle de regroupement des sélections)

Justification de l'algorithme

- Idée de base: réduire la taille des données traitées le plus tôt possible
- Heuristique: réaliser les sélections d'abord, pour réduire la taille des données
  - On réalise d'abord les sélections, car c'est l'opérateur le plus "réducteur"
  - On réalise les jointures (opération très coûteuse) une fois que la taille des données a été réduite au maximum
- Dans le cas général
  - *Plusieurs heuristiques* permettant d'obtenir des plans candidats équivalents
  - *Modèle de coût* d'un plan: estimation du nombre de E/S
  - Choix du meilleur plan candidat suivant le modèle de coût

Résultat après la restructuration



Exemple d'optimisation qui échoue

- Question: le plan ainsi obtenu est-il toujours optimal?
  - Réponse: NON, d'autres facteurs peuvent intervenir
- On rajoute une table *Film*, en plus de *Cinéma*, *Salle*, *Séance*
  Film (*film*, réalisateur, année)
- Requête: les réalisateurs des films qu'on peut voir après 14h

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

- Expressions algébrique
  - Initiale: $\pi_{réalisateur}$ ($\sigma_{heure\text{-}début > 14}$ (***Film*** $\rhd\lhd$ ***Séance***))
  - Optimisée: $\pi_{réalisateur}$ (***Film*** $\rhd\lhd$ $\sigma_{heure\text{-}début > 14}$ (***Séance***))
- Hypothèses
  - *Film* occupe 8 pages et ne contient que 20% des films de *Séance*
  - *Séance* occupe 50 pages et 90% des séances sont après 14h

Exemple (suite)

- Plan initial: $\pi_{réalisateur}(\sigma_{heure\text{-}début > 14}(\textbf{\textit{Film}} \rhd\lhd \textbf{\textit{Séance}}))$
  - Jointure: on lit 8 * 50 = 400 pages et on produit 20% * 50 = 10 pages
  - Sélection: on produit 90% * 10 = 9 pages de séances après14h
  - On laisse de côté la projection (même coût dans les deux cas)

  Coût (E/S): 400E + 10S + 10E + 9S = 429 E/S
- Plan optimisé: $\pi_{réalisateur}(\textbf{\textit{Film}} \rhd\lhd \sigma_{heure\text{-}début > 14}(\textbf{\textit{Séance}}))$
  - 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

Conclusions réécriture algébrique

- La réécriture algébrique est nécessaire, mais pas suffisante
- Il faut tenir compte d'autres critères:
  - *Les chemins d'accès* aux données (selon l'organisation physique)
    - On peut accéder aux données d'une table par accès séquentiel, par index, par hachage, etc.
  - *Les différents algorithmes* possibles pour réaliser un opérateur
    - Il existe par exemple plusieurs algorithmes pour la jointure
    - Souvent 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
- *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 $[v_1,v_2]$ de A → liste d'adresses des articles ayant $A \in [v_1,v_2]$
  - Pour chaque index sur une liste d'attributs $(A_1, A_2, \ldots, A_n)$
    - Valeurs v_i de A_i $(1 \leq i \leq n)$ → liste d'adresses des articles ayant $A_i=v_i$
    - Remarque: un index sur $(A_1, A_2, \ldots, A_n)$ est utilisable aussi comme index sur $(A_1, A_2, \ldots, A_k)$, $k < n$
- *Accès par hachage*
  - Si la table est organisée par hachage sur un attribut A:
    - Étant donnée une valeur v de A → les articles ayant $A=v$

Algorithmes de jointure

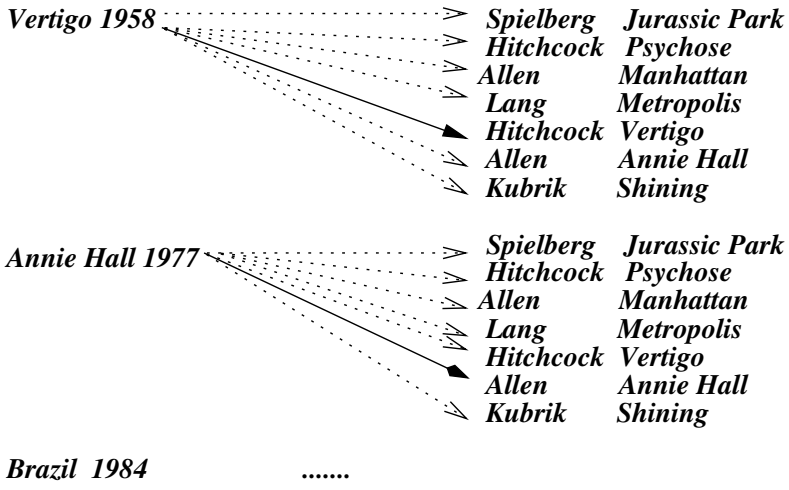
- La jointure est l'opération la plus coûteuse
  - Son optimisation est très importante
- Plusieurs algorithmes, dépendants du chemin d'accès
  - Chacun peut être meilleur dans des situations spécifiques
  - Choix entre plusieurs algorithmes → meilleure optimisation
- Principaux algorithmes
  - Boucles imbriquées simples
  - Tri-fusion
  - Jointure par hachage
  - Boucles imbriquées avec index

Jointure par boucles imbriquées

Algorithme *boucles-imbriquées*
  Entrées: tables R, S
  Sortie: table de jointure J
début
  $J := \emptyset$
  pour chaque r **dans** R **répéter**
    pour chaque s **dans** S **répéter**
      si r joignable à s **alors** $J := J \cup \{r \rhd\lhd s\}$
    fin répéter
  fin répéter
fin

Exemple de jointure par boucles imbriquées

- ***Film*** (titre, année) $\rhd\lhd$ ***Réalisateur*** (nom, titre)



Analyse de coût

- La lecture des articles d'une table dans une boucle:
  - On lit sur disque les *pages* de la table, pour les charger en mémoire
  - On lit en mémoire les *articles* de chaque page
- Pour le coût: seule compte la lecture des pages sur disque
- Hypothèses
  - Disque: T_R pages pour R, T_S pages pour S
  - Mémoire pour la lecture de R, S : 2 pages (une page pour chacune)
- Coût: $T_R + T_R \times T_S = \mathcal{O}(T_R \times T_S)$
  - Pour chaque page de R on lit toutes les pages de S et on fait la jointure page par page entre les articles
    - S est lue T_R fois et R une seule fois
  - On ne tient pas compte du coût de l'écriture du résultat, car il sera le même quel que soit l'algorithme de jointure

Analyse de coût (suite)

- Que se passe-t-il si l'on dispose de M pages de mémoire?
  - Supposons qu'on alloue un tampon de K pages à R et le reste à S
  - Pour chaque lecture du tampon de R il faut lire S en entier
    - → On lit $\lceil T_R/\mathrm{K} \rceil$ fois la table S et une seule fois la table R
    - → La lecture de S n'a besoin que d'un minimum de mémoire (1 page)
  - Donc on a besoin d'allouer le maximum de mémoire à R (M-1 pages) et le minimum à S (1 page)
- Coût: $T_R + \lceil T_R/(\mathrm{M}\text{-}1) \rceil \times T_S$
  - → Si R tient en mémoire alors le coût est $T_R + T_S$!
- Conclusions
  - La jointure par boucles imbriquées est *inefficace* sur de grandes tables
  - Toutefois, si l'une des relations entre en mémoire → elle est *très efficace*

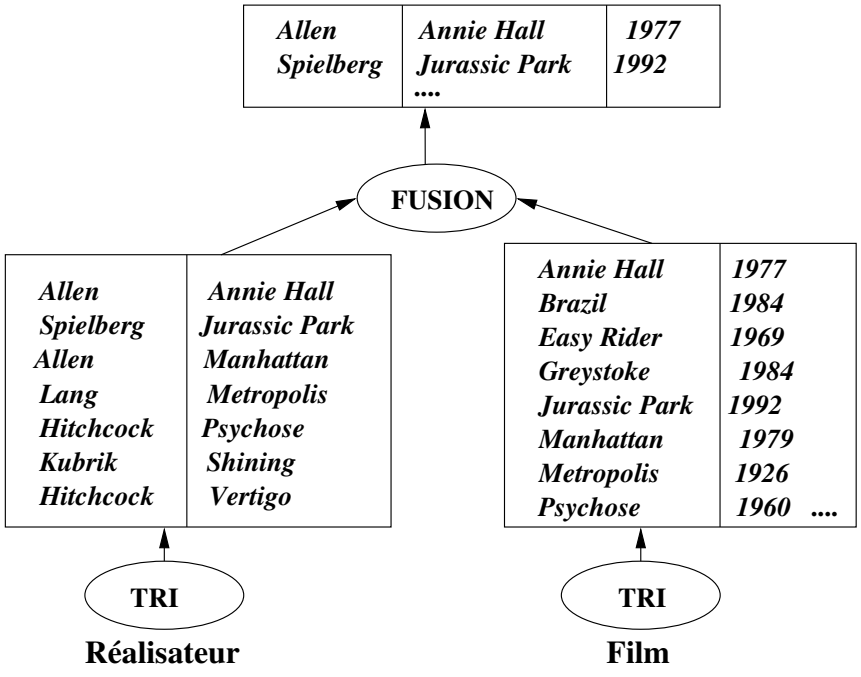
Jointure par tri-fusion

Algorithme *tri-fusion*
Entrées: tables R, S
Sortie: table de jointure $J = R \rhd\lhd_{R.A=S.B} S$
début
trier R sur l'attribut de jointure A
trier S sur l'attribut de jointure B
$J :=$ ***fusion*** (R, S)
fin

- *Fusion*: on parcourt en parallèle R et S, en joignant les articles $r \in R$ et $s \in S$ qui ont $r.A = s.B$

Exemple de jointure par tri-fusion

- ***Réalisateur*** (nom, titre) $\rhd\lhd$ ***Film*** (titre, année)



Fusion

- Si l'une des tables a des valeurs distinctes pour l'attribut de jointure
  → on la place en seconde position (table S)

Algorithme *fusion*
  Entrées: tables R triée sur A, S triée sur B
  Sortie: table de jointure $J = R \rhd\lhd_{R.A=S.B} S$
début
  $J := \emptyset$; $r :=$ premier(R); $s :=$ premier(S)
  tant que r existe **et** s existe **répéter**
    si $r.A = s.B$ **alors**
      $J := J \cup \{r \rhd\lhd_{R.A=S.B} s\}$
      $r :=$ suivant(R)
    sinon si $r.A < s.B$ **alors** $r :=$ suivant(R)
    sinon $s :=$ suivant(S)
  fin répéter
fin

Fusion (suite)

- L'avantage du cas précédent: on ne revient jamais en arrière, donc un seul parcours des deux tables
- Si les deux tables ont des doublons pour l'attribut de jointure
  - Pour chaque cas où $r.A = v$ (n fois) et $s.B = v$ (m fois), il faut produire $n \times m$ articles dans la jointure
  - A chaque nouvel article r, il faut prendre tous les s tel que $s.B = r.A$ → il faut revenir en arrière dans S → similaire aux boucles imbriquées
  - Comportement similaire à la fusion simple + boucles imbriquées locales (là où il y a des doublons dans R et S pour une même valeur d'attribut de jointure)

Analyse de coût tri-fusion

- Tri: pour une relation R avec M pages de mémoire
  - Un passage de tri en mémoire → $\lceil T_R/M \rceil$ séquences de taille M
  - $\log_{M-1}(\lceil T_R/M \rceil))$ passages de fusion de M-1 séquences
  - Chaque passage lit et écrit T_R pages

    $\text{Coût}_{tri} = 2\ T_R\ (1+\log_{M-1}(\lceil T_R/M \rceil)) = \mathcal{O}\ (T_R \times \log(T_R))$
- Fusion
  - Fusion simple: un seul parcours de chaque relation

    $\text{Coût}_{fusion} = T_R + T_S$
  - Fusion avec doublons: en moyenne un peu moins bon
    - Au pire tous les articles ont la même valeur pour l'attribut de jointure → coût des boucles imbriquées: $\text{Coût}_{fusion} = T_R \times T_S$
- Coût total: $2\ T_R\ (1+\log_{M-1}(\lceil T_R/M \rceil)) + 2\ T_S\ (1+\log_{M-1}(\lceil T_S/M \rceil)) + T_R + T_S$

  → Le tri est plus coûteux que la fusion

  → *Le coût de la jointure par tri-fusion est donné essentiellement par le tri*

Comparaison boucles imbriquées – tri fusion

- Pour des grandes relations → tri fusion plus efficace

  Ex. $T_R = T_S = 1000$ pages

  $\text{Coût}_{bi} \approx T_R \times T_S = 1.000.000$ lectures de pages

  $\text{Coût}_{tf} \approx T_R \times \log(T_R) + T_S \times \log(T_S) \approx 1000*10 + 1000*10 = 20.000$
  - Autre avantage du tri-fusion: élimination des doublons, groupement, affichage ordonné *plus rapides*
- Si l'une des relations entre en mémoire → boucles imbriquées plus efficace

  Ex. $T_R = 30$ pages, $T_S = 1000$ pages

  $\text{Coût}_{bi} \approx T_R + T_S = 1030$ lectures de pages

  $\text{Coût}_{tf} \approx T_R \times \log(T_R) + T_S \times \log(T_S) \approx 30*5 + 1000*10 > 10.000$

Jointure par hachage

- Même principe que les boucles imbriquées avec M pages mémoire
  - Chaque M-1 pages de R lues → table de hachage en mémoire
  - Jointure avec chaque ligne de S plus rapide en mémoire
- Coût lecture/écriture disque: identique aux boucles imbriquées

  $\text{Coût}_{jh} = T_R + \lceil T_R/(M-1) \rceil \times T_S$
  - Si R entre en mémoire → $\text{Coût}_{jh} = T_R + T_S$
- Jointure par hachage: un peu plus efficace que les boucles imbriquées
  - Variantes plus efficaces si les deux tables sont de grande taille
- Inconvénient hachage: seulement pour la *jointure avec égalité*
  - Les boucles imbriquées et même le tri-fusion applicables aux jointures avec inégalité

Jointure avec une table indexée

Algorithme *boucles-imbriquées-index*
  Entrées: tables R, S; index sur $S.B$
  Sortie: table de jointure $J = R \rhd\lhd_{R.A=S.B} S$
début
  $J := \emptyset$
  pour chaque $r \in R$ **répéter**
    pour chaque $s \in Index_{S.B}(r.A)$ **répéter**
      $J := J \cup \{r \rhd\lhd_{R.A=S.B} s\}$
    fin répéter
  fin répéter
fin

- Fonction $Index_{S.B}(r.A)$: cherche à l'aide de l'index les articles de S dont l'attribut B a pour valeur $r.A$
  - Généralement l'index retourne une liste d'adresses d'articles (ROWID), utilisées pour obtenir chaque article de S

Analyse de coût jointure avec index

- Hypothèses
  - Index B+, stockant les adresses des articles
  - k = nb. moyen d'entrées dans une feuille de l'index
  - *Sélectivité* de l'attribut $S.B$: $\lambda_{S.B}$ = nombre de valeurs distinctes de $S.B$ / $|S|$
    - $1/\lambda_{S.B}$ = le nombre moyen d'articles ayant une même valeur pour $S.B$
- Coût $Index_{S.B}\,(r.A)$ = coût recherche index + coût lecture articles
  - Coût recherche index = coût recherche 1ère feuille + coût parcours feuilles
    - Coût recherche 1ère feuille = $\mathcal{O}\,(\log(|S|))$
    - Coût parcours feuilles = nb. feuilles ayant la valeur recherchée = $\lceil 1 / k * \lambda_{S.B} \rceil$
  - Coût lecture articles: pour chaque adresse de l'index il faut lire une page
    - Coût lecture articles = $\lceil 1 / \lambda_{S.B} \rceil$

  → Coût $Index_{S.B}\,(r.A) = \mathcal{O}\,(\log(|S|) + 1 / \lambda_{S.B})$

Analyse de coût (suite)

- Le facteur sélectivité
  - Index unique → $\lambda_{S.B}$=1, donc le facteur est négligeable (une page de plus)
  - Index non-unique → $1 / \lambda_{S.B}$ peut devenir prédominant (au max. |S| !)
- Coût jointure avec index: pour chaque $r \in R$ on accède l'index
  - Coût total = $T_R + |R| \times$ coût $Index_{S.B}\,(r.A)$

→ Coût total = $\mathcal{O}\,(|R| \times (\log(|S|) + 1 / \lambda_{S.B}))$

- Facteur sélectivité
  - Index unique ou très sélectif → $\mathcal{O}\,(|R| \times \log(|S|))$
  - Mauvaise sélectivité (ex. p valeurs seulement) → $\mathcal{O}\,(|R| \times |S| / p)$
- Conclusion : efficace seulement si l'index est très sélectif

Statistiques

- Importance des statistiques
  - La taille des relations permet de choisir entre les algorithmes de jointure
  - La sélectivité des attributs permet de juger de l'opportunité de l'utilisation d'un index
  - etc.
- Module d'acquisition de statistiques sur la base
  - Une possibilité: déclenchement périodique
  - Une autre variante: estimation en temps réel par échantillonnage

Algèbre physique

- Plan d'exécution algébrique
  - Plusieurs chemins d'accès possibles vers les données
  - Plusieurs algorithmes possibles pour un opérateur algébrique

→ On a besoin d'une algèbre "plus fine" qui exprime des chemins d'accès et des opérations intermédiaires

- Algèbre physique
  - Chemins d'accès aux données
  - Opérations physiques

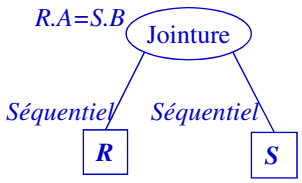
Opérateurs de l'algèbre physique

CHEMINS D'ACCES		*OPERATIONS PHYSIQUES*	
Sequentiel TABLE	*Parcours sequentiel*	Critere Selection	*Selection selon un critere*
		Critere Filtre	*Filtre d'un ensemble en fonction d'un autre*
Adresse TABLE	*Acces par adresse*	Attribut(s) Tri	*Tri sur un attribut*
		Critere Jointure	*Jointure selon un critere*
Attribut(s) INDEX	*Parcours d'index*	Critere Fusion	*Fusion de deux ensembles tries*
		Attribut(s) Projection	*Projection sur des attributs*

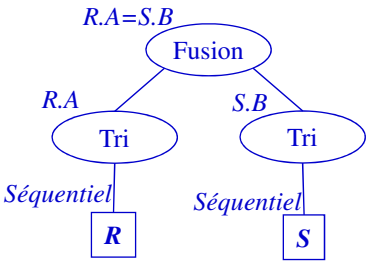
Exemples: passage algèbre logique - physique

- **Jointure** ($R \triangleright\!\triangleleft_{R.A=S.B} S$): en fonction de l'algorithme

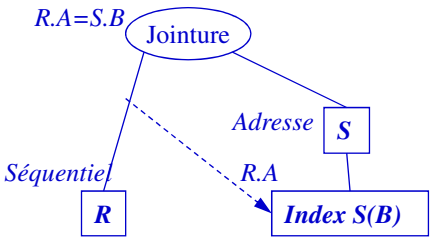
Boucles imbriquées



Tri – fusion

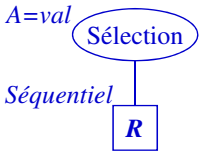


Avec index

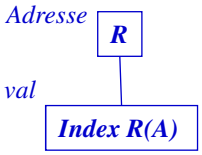


- **Sélection** ($\sigma_{A=val}(R)$)

Directe



Avec index

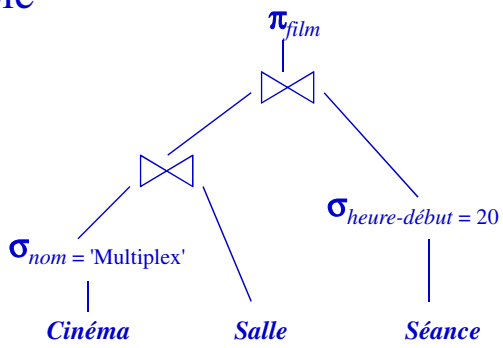


Exemple de plan physique

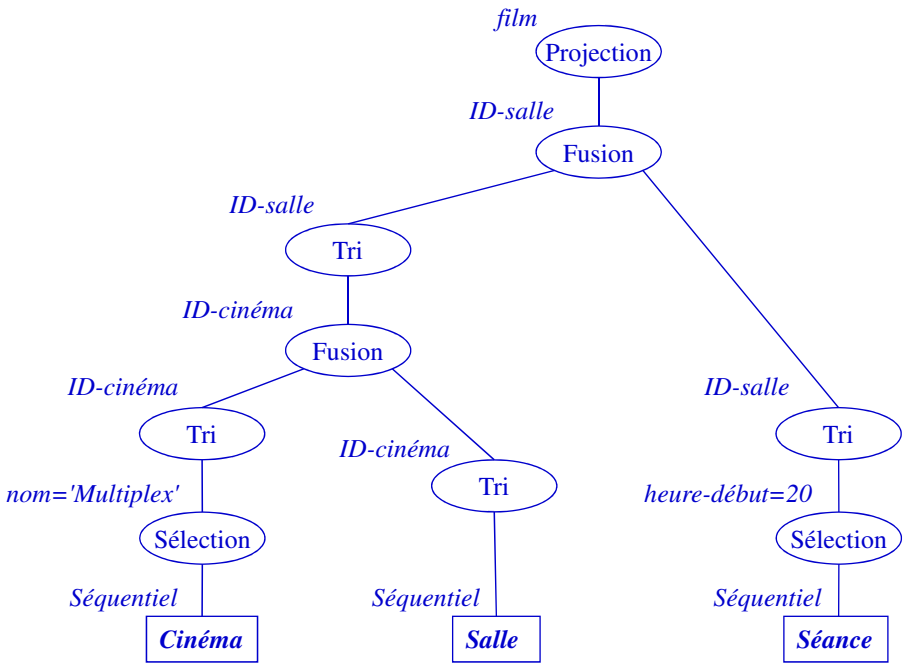
- Requête: quels films passent au Multiplex à 20 heures?

```
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
```

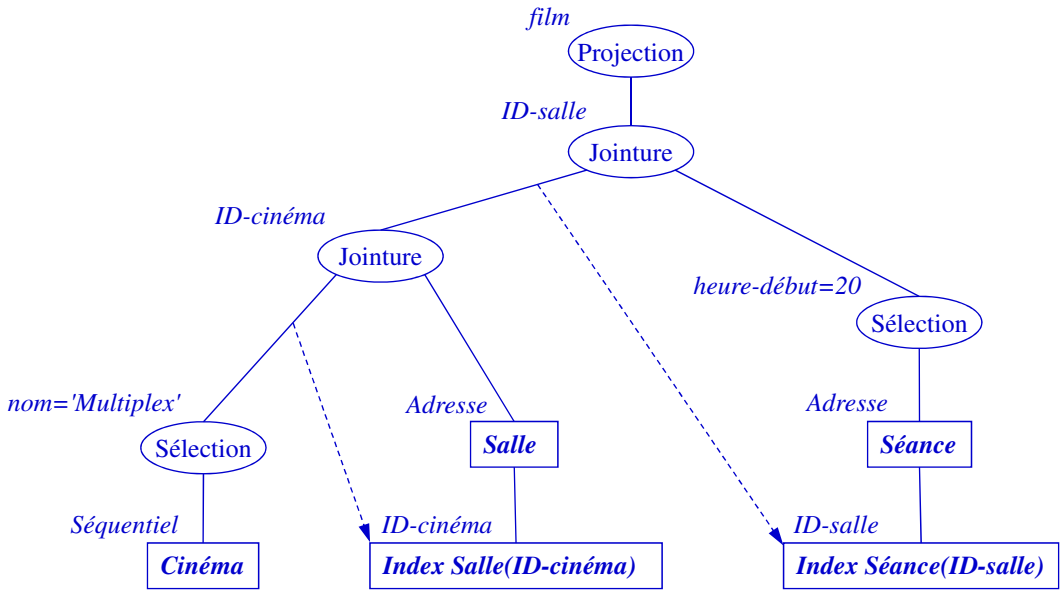
- Plan logique possible



Plan physique sans index, ni hachage



Plan physique avec index sur les attributs de jointure



Évaluation de requêtes

- Optimisation → plan d'exécution physique
- Évaluation: exécution de ce plan
  - Accès aux données
  - Exécution des algorithmes des opérateurs du plan
  - Gestion des flots de données entre opérations
- Objectif: minimiser le nombre de lectures/écritures de pages
- Approche la plus utilisée: évaluation itérative en pipeline
  - Résultats produits au fur et à mesure (opérateurs = itérateurs)
  - Efficace: stockage données intermédiaires réduit au minimum
  - Obtention rapide des premiers résultats → applications interactives

Techniques d'accès aux données

- Parcours séquentiel: systématiquement optimisé dans les SGBD
- Principales techniques
  - Regroupement des pages sur des espaces contigus
    - "Extensions" dans Oracle
  - Lecture en avance: à la lecture d'une page on lit également les *n* suivantes
    - Typiquement *n*=7 ou *n*=15
- Conséquence
  - On lit les pages par blocs contigus, ce qui est plus rapide que la lecture successive des pages

Utilisation d'un tampon

- Tampon ("buffer", cache): zone de mémoire qui permet de stocker des pages
- Problème complexe: essayer de garder dans le tampon les pages susceptibles d'être réutilisées "prochainement"
- Principe: utilisation d'un *gestionnaire du tampon*
  - Un programme exécutant une requête ne demande pas directement la lecture/écriture d'un page, mais s'adresse au gestionnaire du tampon
  - Le gestionnaire vérifie que la page se trouve dans le tampon (sinon il la lit du disque) avant de réaliser l'opération
- Types de pages
  - *Statiques*: c'est le programme qui demande au gestionnaire de la libérer
  - *Volatiles*: la page est à la disposition du gestionnaire

Parcours d'index

- La plupart des SGBD utilisent des variantes de l'arbre B
- Utilisation pour les opérations pour lesquelles l'index est conçu
  - Recherche par clé
  - Recherche par intervalle de clés
- Optimisation d'autres opérations en utilisant l'index
  - Éviter l'accès aux articles si le(s) champ(s) recherchés sont dans l'index
  - Compter les articles qui respectent une condition liée à la clé de l'index
    - Si la table SEANCE est indexée sur l'heure de début, on peut répondre à la question « Nombre de séances après 21h » sans consulter la table
  - Test d'existence sur une condition liée à la clé de l'index
    - Idem pour la requête « Y a-t-il des séances après 22h? »

Parcours d'index avec tampon

- Exemple: soit la requête suivante, en supposant un index sur *année*
  SELECT titre FROM Film WHERE année IN (1956, 1934, 1992, 1997)
- Évaluation simple: pour chaque valeur du IN on trouve dans l'index les adresses d'articles et on lit pour chaque adresse l'article pour récupérer le titre
- Évaluation optimisée:
  - On cherche dans l'index les adresses pour *toutes* les valeurs du IN
  - On groupe ces adresses par numéro de page
  - On lit chaque page et on extrait les articles et leur champ titre
  - → Avantage: on lit chaque page d'articles *une seule fois*
  - → Inconvénient: on doit attendre la récupération de toutes les adresses de l'index avant de calculer des résultats
- Variante intermédiaire: tampon pour accumuler les adresses
  - Regroupement par page quand le tampon est plein

Évaluation des opérateurs

- Types d'opérateurs
  - Pipeline: qui peuvent calculer des résultats un par un, en demandant une par une les valeurs d'entrée
  - Non-pipeline: qui ont besoin de toutes les données d'entrée pour pouvoir produire des résultats
- Avantage des opérateurs pipeline
  - Peuvent produire rapidement des résultats, adaptés à des applications interactives
  - Pas besoin de stocker des données intermédiaires, car chaque résultat d'un opérateur peut être consommé tout de suite par un autre opérateur
    → gain de performances
- Les opérateurs de l'algèbre physique présentée:
  - Tous sont pipeline sauf le tri

Implémentation des opérateurs

- Implémentation sous forme d'*itérateurs*
  - Itérateur: objet qui à chaque appel produit le résultat suivant
  - Objectif: produire des résultats au fur et à mesure
    - Un plan composé d'itérateurs est lui-même un itérateur
  - Compatibles à la fois avec les opérateurs pipeline et non-pipeline
  - Principales opérations sur un itérateur: *initialisation*( ), *suivant*( )
- Exemple
  - Projection globale

    pour chaque article *a* en entrée **répéter**
    résultat ← *résultat* ∪ *projection*(*a*)
    retourner *résultat*
  - Itérateur projection (opération *suivant()*)

    a = entrée.*suivant*()
    retourner *projection*(*a*)

Exemple d'itérateurs

- Sélection.*suivant*()

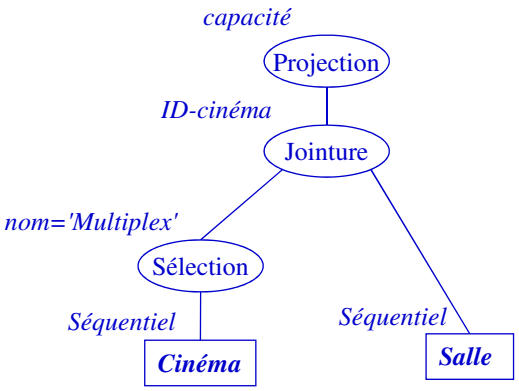
répéter
  a ← entrée.*suivant*()
  si *a* = *nil* **alors retourner** *nil*
  si *condition*(*a*) **alors retourner** *a*
fin répéter

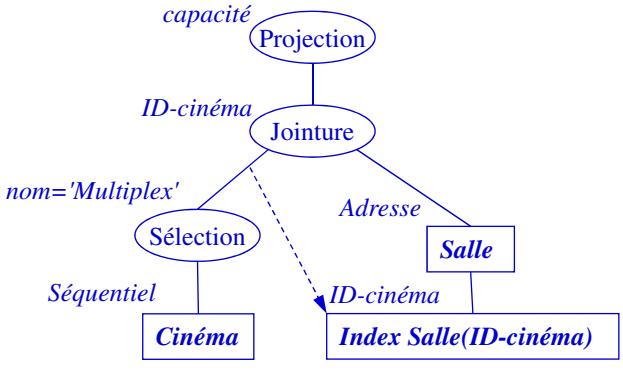
- Jointure.*suivant*()
  - Contexte: *g*, *d* = dernière entrée à gauche/droite; *init*() initialise *g*

si *g* = *nil* **alors retourner** *nil*
répéter
  d ← entrée2.*suivant*()
  si *d* = *nil* **alors** *g* ← entrée1.*suivant*()
    si *g* = *nil* **alors retourner** *nil*
    d ← entrée2.*init*()
  fin si
  si *condition*(*g*,*d*) **alors retourner** *g* ▷◁ *d*
fin répéter

Exécution d'un plan

- On demande le résultat suivant à l'opérateur racine
  - Celui-ci demande le résultat suivant à ses entrées, etc.
- Propagation des commandes racine → feuilles
- Propagation des données feuilles → racine





Optimisation dans Oracle

- Approche classique
  - Génération de plusieurs plans d'exécution physiques
  - Estimation du coût de chaque plan
  - Choix du meilleur plan et exécution
- Algèbre physique
  - Chemins d'accès aux données: séquentiel, index, hash, cluster
  - Opérateurs de traitement: boucles imbriquées, filtre, tri, fusion, …
- Outils
  - EXPLAIN: visualisation des plans d'exécution
  - ANALYSE: production de statistiques
  - TKPROF: mesure du temps d'exécution

Chemins d'accès aux données dans Oracle

- Parcours séquentiel
  - TABLE ACCESS FULL
- Accès direct par adresse
  - TABLE ACCESS BY (INDEX|USER|…) ROWID
- Accès par index
  - INDEX (UNIQUE|RANGE|…) SCAN
- Accès par hachage
  - TABLE ACCESS HASH
- Accès par cluster
  - TABLE ACCESS CLUSTER

Opérateurs physiques

- Pour la jointure
  - Boucles imbriquées: NESTED LOOPS
  - Tri-fusion: SORT JOIN, MERGE JOIN
  - Hachage: HASH JOIN
- Autres opérations
  - Union d'ensembles d'articles: CONCATENATION, UNION
  - Intersection d'ensembles d'articles: INTERSECTION
  - Différence d'ensembles d'articles: MINUS
  - Filtrage d'articles d'une table basé sur une autre table: FILTER
  - Intersection d'ensembles de ROWID: AND-EQUAL
  - …

Plan d'exécution EXPLAIN

- Commande
  - EXPLAIN PLAN FOR
    SELECT… FROM… WHERE…
- Description textuelle du plan structurée sous forme de table (PLAN_TABLE)
  - Chemins d'accès
  - Opérateurs physiques
  - Ordre des opérateurs: structure d'arbre

Exemple

- Schéma relationnel

  CINEMA (ID-cinéma*, Nom, Adresse)
  SALLE (ID-salle, Nom, Capacité+, ID-cinéma+)
  FILM (ID-film, Titre, Année, ID-réalisateur+)
  SEANCE (ID-séance*, Heure-début, Heure-fin, ID-salle+, ID-film)
  ARTISTE (ID-artiste*, Nom, Date-naissance)

  * : index unique sur l'attribut
  + : index non-unique sur l'attribut

Exemple (suite)

- Requête: les films qui commencent à 20h au Multiplex

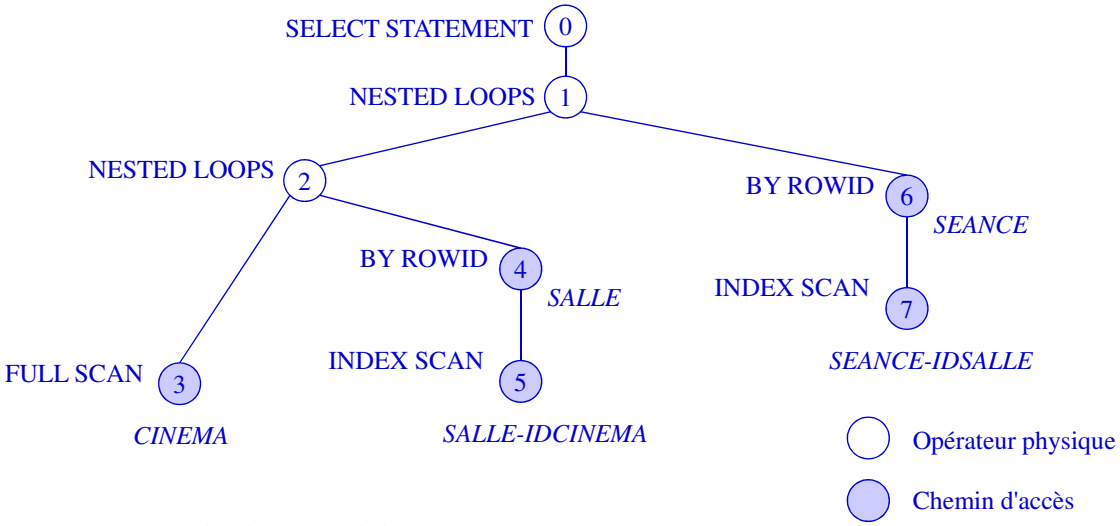
```
SELECT ID-film
FROM CINEMA, SALLE, SEANCE
WHERE CINEMA.ID-cinéma = SALLE.ID-cinéma AND
        SALLE.ID-salle = SEANCE.ID-salle AND
        CINEMA.nom = 'Multiplex' AND
        SEANCE.Heure-début = 20
```

- Plan d'exécution produit par EXPLAIN

```
0  SELECT STATEMENT
  1  NESTED LOOPS
    2  NESTED LOOPS
      3  TABLE ACCESS FULL CINEMA
      4  TABLE ACCESS BY INDEX ROWID SALLE
        5  INDEX RANGE SCAN SALLE-IDCINEMA
    6  TABLE ACCESS BY INDEX ROWID SEANCE
        5  INDEX RANGE SCAN SEANCE-IDSALLE
```

Exemple (suite)

- Remarque: certaines opérations ne sont pas représentées
  - *Ex.* Les sélections sont considérées automatiquement appliquées sur l'accès aux articles de la table
- Représentation sous forme d'arbre du plan EXPLAIN



Exemples de plans d'exécution

- Sélection sans index

  SELECT * FROM CINEMA
  WHERE nom = 'Multiplex'

  – Plan d'exécution

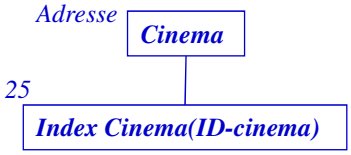
  0 SELECT STATEMENT
  1 TABLE ACCESS FULL CINEMA

- Sélection avec index

  SELECT * FROM CINEMA
  WHERE ID-cinéma = 25

  – Plan d'exécution

  0 SELECT STATEMENT
  1 TABLE ACCESS BY INDEX ROWID CINEMA
  2 INDEX UNIQUE SCAN CINEMA-IDCINEMA

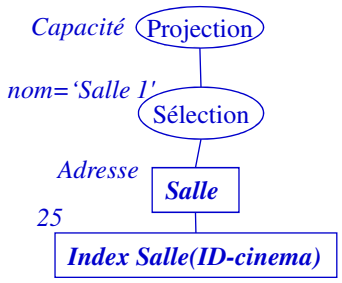


Sélection multiple

- Sélection conjonctive avec un index

  SELECT Capacité FROM SALLE
  WHERE ID-cinéma = 25 AND Nom = 'Salle 1'

- Plan d'exécution
  - 0 SELECT STATEMENT
    1 TABLE ACCESS BY INDEX ROWID SALLE
      2 INDEX RANGE SCAN SALLE-IDCINEMA

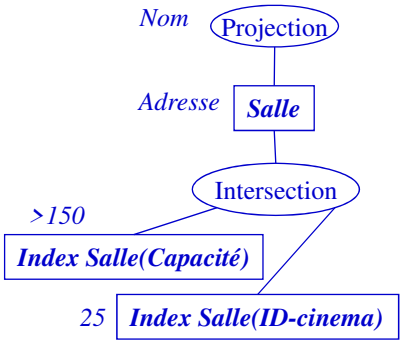


- Sélection conjonctive avec deux index

  SELECT Nom FROM SALLE
  WHERE ID-cinéma = 25 AND Capacité > 150

- Plan d'exécution

  0 SELECT STATEMENT
    1 TABLE ACCESS BY INDEX ROWID SALLE
      2 AND-EQUAL
        3 INDEX RANGE SCAN SALLE-IDCINEMA
        4 INDEX RANGE SCAN SALLE-CAPACITE



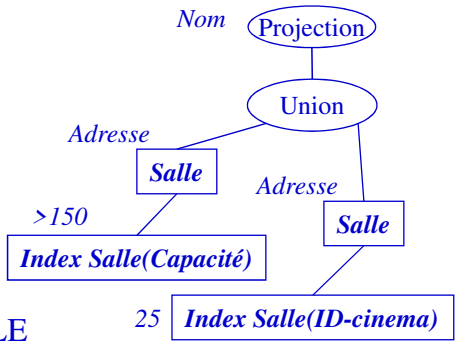
Sélection disjonctive

- Sélection disjonctive avec index

  SELECT Nom FROM SALLE
  WHERE ID-cinéma = 25 OR Capacité > 150

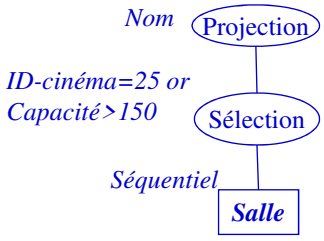
- Plan d'exécution

  0 SELECT STATEMENT
    1 CONCATENATION
      2 TABLE ACCESS BY INDEX ROWID SALLE
        3 INDEX RANGE SCAN SALLE-CAPACITE
      4 TABLE ACCESS BY INDEX ROWID SALLE
        5 INDEX RANGE SCAN SALLE-IDCINEMA



- Pour la même requête, un plan alternatif

  0 SELECT STATEMENT
    1 TABLE ACCESS FULL SALLE



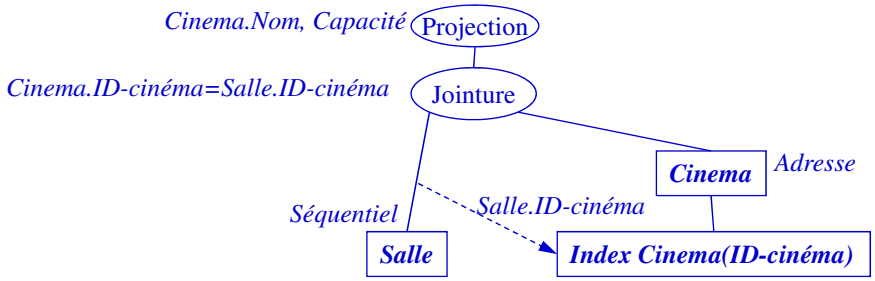
Jointure avec index

- Requête

  SELECT CINEMA.Nom, Capacité
  FROM CINEMA, SALLE
  WHERE CINEMA.ID-cinéma = SALLE.ID-cinéma

  – Plan d'exécution

  0 SELECT STATEMENT
  1 NESTED LOOPS
  2 TABLE ACCESS FULL SALLE
  3 TABLE ACCESS BY INDEX ROWID CINEMA
  4 INDEX UNIQUE SCAN CINEMA-IDCINEMA



Jointure sans index

- Requête

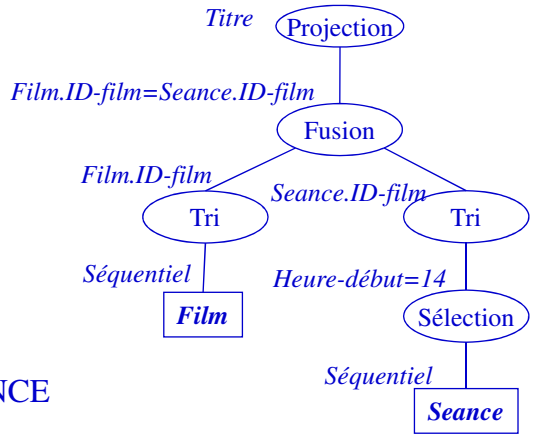
  SELECT Titre
  FROM FILM, SEANCE
  WHERE Heure-début = 14 AND
  FILM.ID-film = SEANCE.ID-film

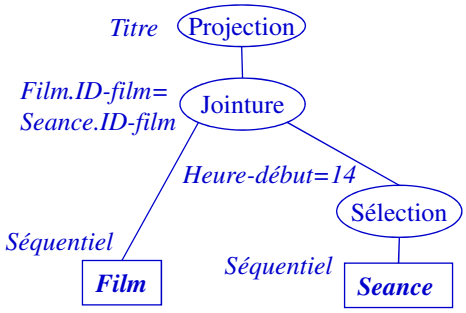
  – Plan d'exécution

  0 SELECT STATEMENT
  1 MERGE JOIN
  2 SORT JOIN
  3 TABLE ACCESS FULL SEANCE
  4 SORT JOIN
  5 TABLE ACCESS FULL FILM

- Plan alternatif (si par exemple FILM tient en mémoire)

  0 SELECT STATEMENT
  1 NESTED LOOPS
  2 TABLE ACCESS FULL FILM
  3 TABLE ACCESS FULL SEANCE

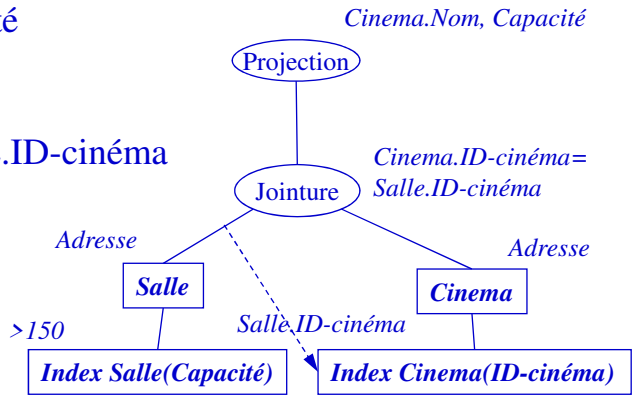




Jointure et sélection avec index

- Requête

```
SELECT CINEMA.Nom, Capacité
FROM CINEMA, SALLE
WHERE Capacité > 150 AND
   CINEMA.ID-cinéma = SALLE.ID-cinéma
```



- Plan d'exécution

```
0  SELECT STATEMENT
  1  NESTED LOOPS
    2  TABLE ACCESS BY INDEX ROWID SALLE
      3  INDEX RANGE SCAN SALLE-CAPACITE
    4  TABLE ACCESS BY INDEX ROWID CINEMA
      5  INDEX UNIQUE SCAN CINEMA-IDCINEMA
```