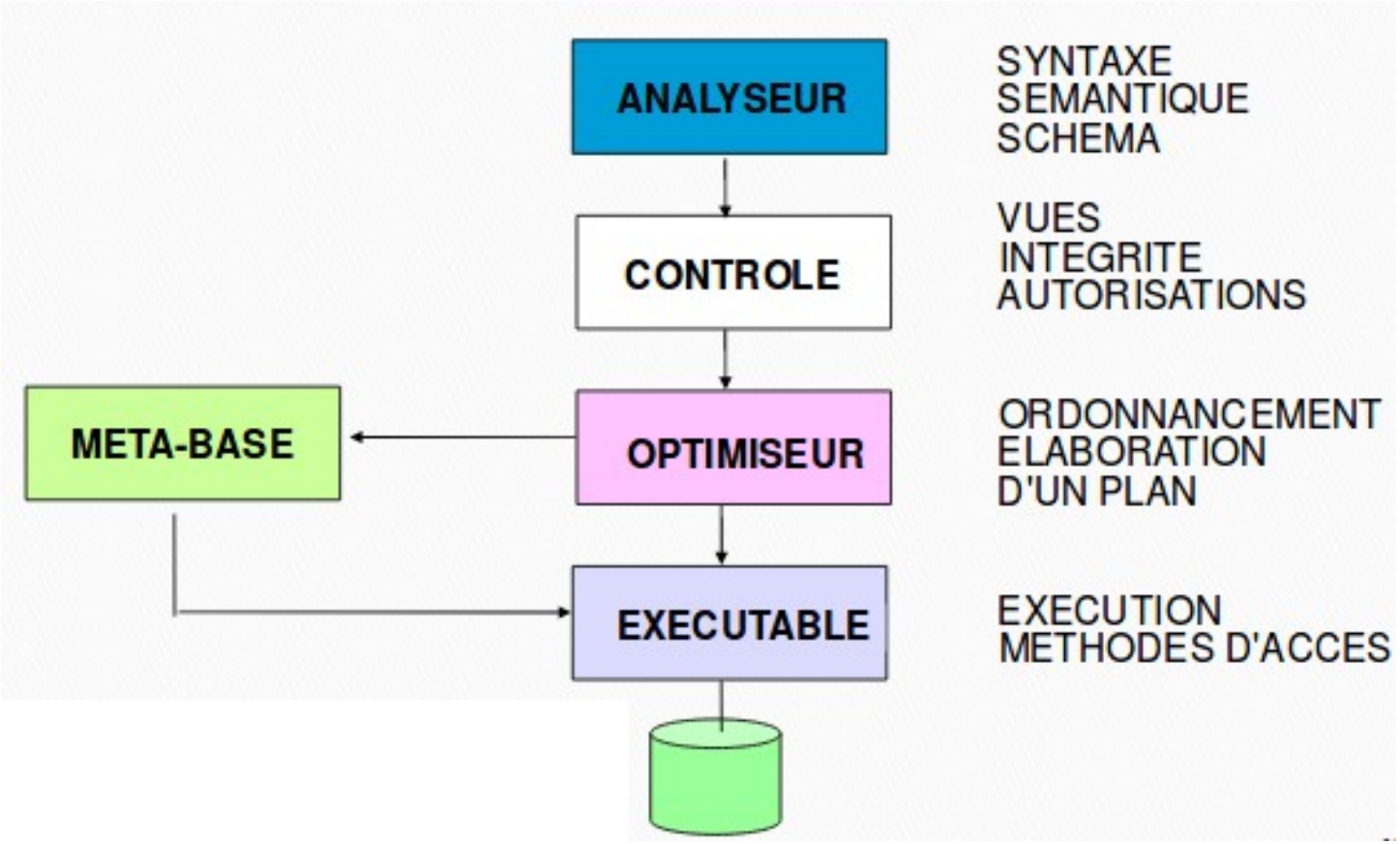
Entrepôts de Données et Big Data - HAI708I

Optimisation de Requête

Référence : cours de Serge Abiteboul

Architecture type d’un SGBD



Pourquoi on s’intéresse à l’optimisation ?

- **Volume de données (>~1000 lignes)**
- **Requêtes consommatrices de temps et de ressources**
- **Optimisation = tâche du SGBD**

- **Mais**
  - requête mal écrite
    ➡ mauvaise optimisation
  - possibilité d’influer sur l’optimisation faite par le SGBD
    ➡ meilleure optimisation

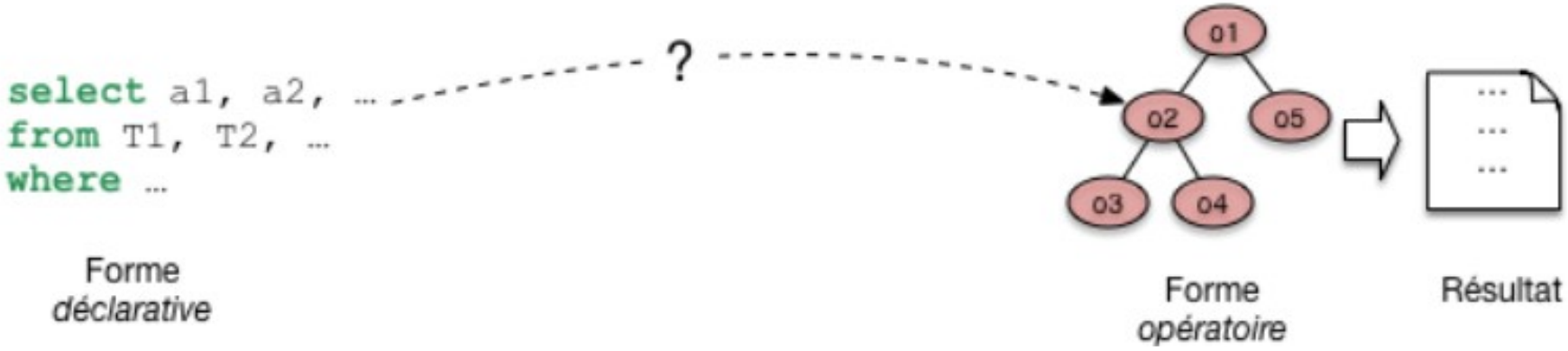
➡ **Nécessité de comprendre les mécanismes de l’optimisation de requêtes**

Les bonnes pratiques pour écrire une requête

- **Index non utilisé si :**
  - fonction ou d’opérateur utilisés sur une colonne indexée
  - comparaison des colonnes indexées avec la valeur null
- **Éviter les opérations inutiles**
  - le select *
  - le tri
  - filtre sur les données le plus tôt possible dans le cas de requêtes imbriquées et de jointures
- **Favoriser les opérations les moins coûteuses**
  - Favoriser les UNION/UNION ALL aux OR
  - Favoriser le EXISTS par rapport au IN lorsque la liste à parcourir est issue d’une sous-requête et pas d’une liste statique
  - Attention au IN/NOT IN lorsqu’il y a des valeurs nulles : il ne peut pas les comparer et considère qu’elles n’existent pas.

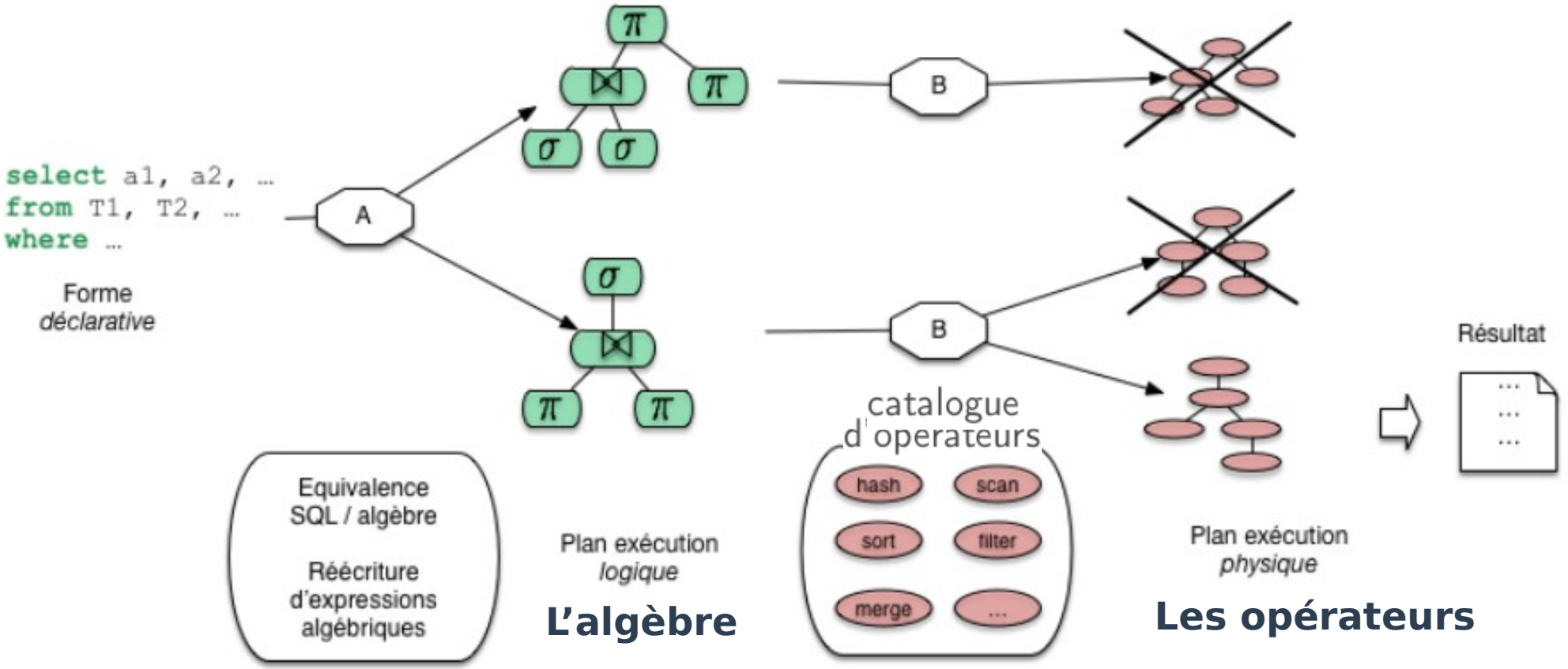
Qu’est-ce que « optimiser » ?

- **Requête SQL déclarative : elle ne dit pas comment calculer le résultat.**
- **Besoin d’un programme : le plan d’exécution = arbre constitué d’opérateurs**



Qu’est-ce que « optimiser » ?

- **Deux étapes pour obtenir le plan d’exécution**



- **À chaque étape, plusieurs choix : le SGBD les évalue et choisit le « meilleur »**

Un exemple

- **Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.**

Requête SQL

```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```

2 sélections
1 jointure
1 projection

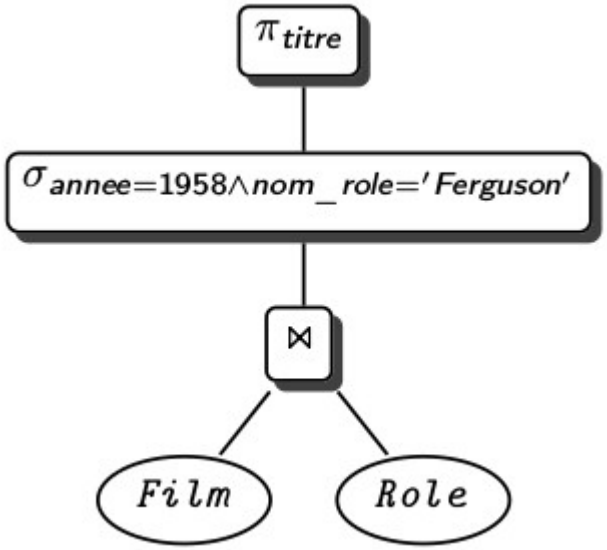
Un exemple

- **Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.**

Requête SQL ➡ **Plan d'exécution logique (l'algèbre)**

$$\pi_{titre}(\sigma_{annee=1958 \wedge nom\_role='\text{Ferguson}'}(Film \bowtie_{id=id\_film} Role))$$

```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```



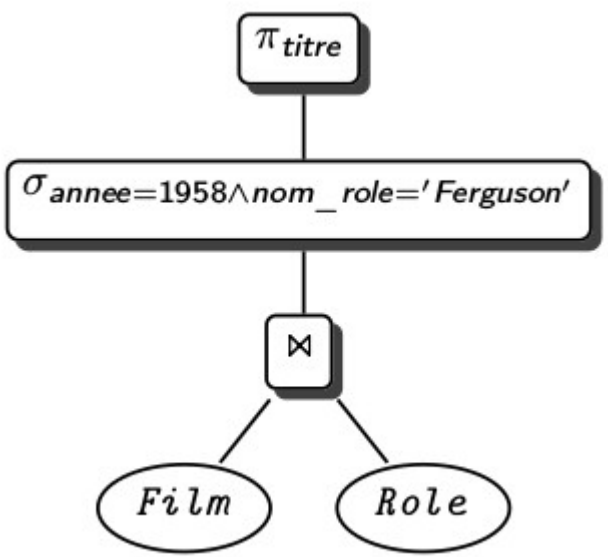
2 sélections
1 jointure
1 projection

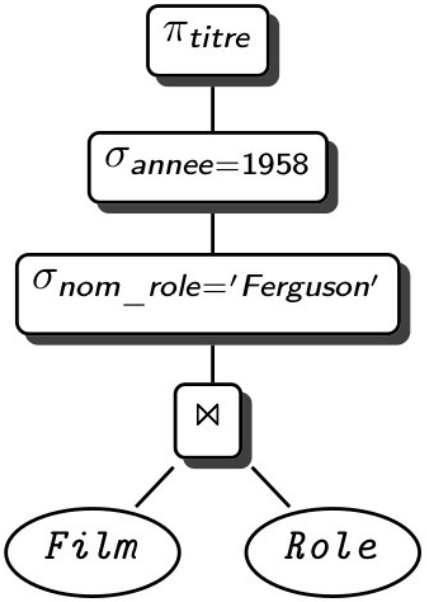
Un exemple

- **Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.**

Requête SQL ➡ **Plan d'exécution logique (l'algèbre)**

```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```



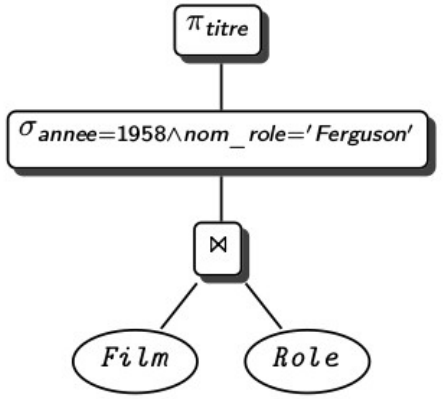


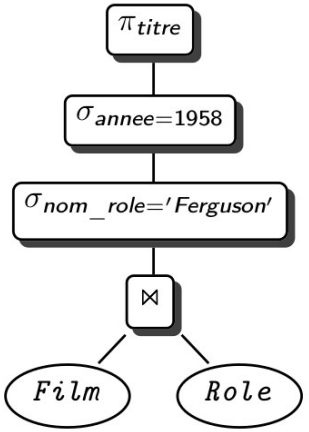
Un exemple

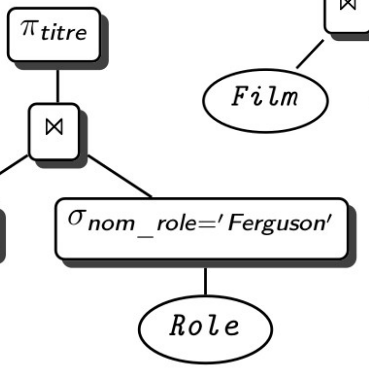
- **Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.**

Requête SQL ➡ **Plan d'exécution logique (l'algèbre)**

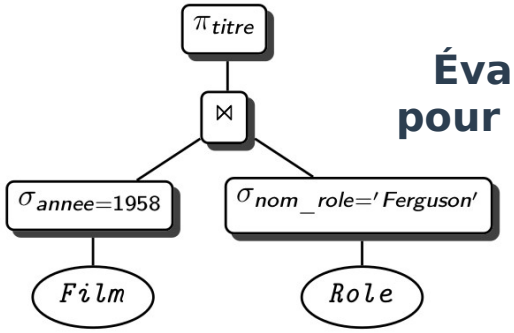
```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```







Plusieurs écritures possibles ⬅ **Évaluation des coûts pour trouver le meilleur plan logique**

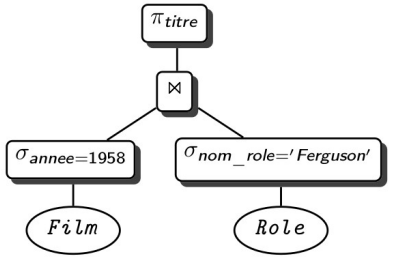


Un exemple

- **Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.**

Requête SQL ➡ **Plan d'exécution logique (l'algèbre)**

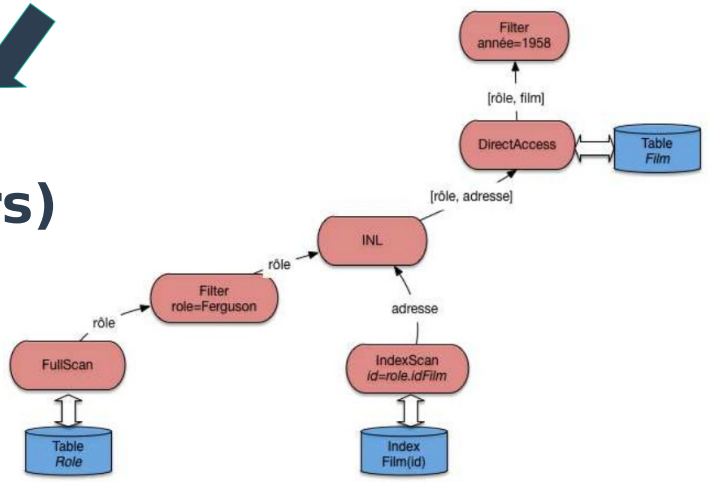
```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```



Plan d'exécution physique (opérateurs)

Un opérateur = une opération

Plusieurs algorithme par opération

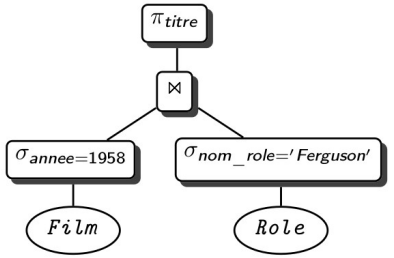


Un exemple

- **Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.**

Requête SQL ➡ **Plan d'exécution logique (l'algèbre)**

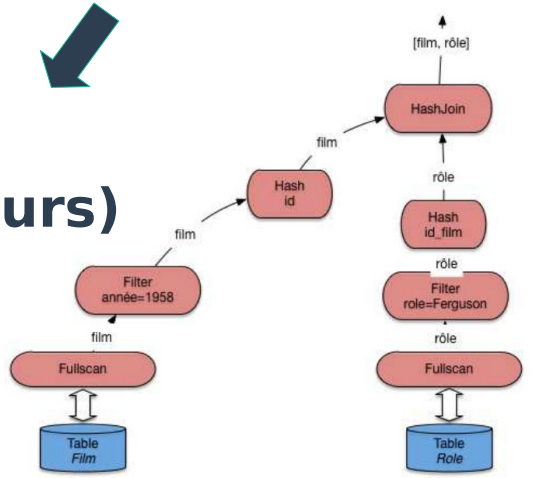
```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```



Plan d'exécution physique (opérateurs)

Un opérateur = une opération

Plusieurs algorithme par opération



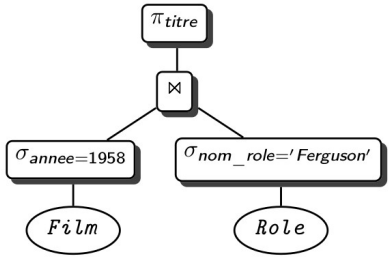
Plusieurs plan possibles

Un exemple

- **Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.**

Requête SQL ➡ **Plan d'exécution logique (l'algèbre)**

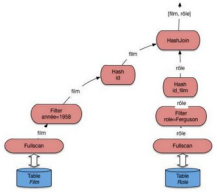
```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```

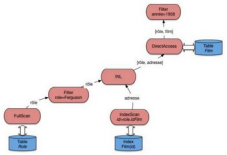


Plan d'exécution physique (opérateurs)

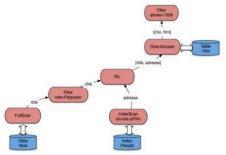
Un opérateur = une opération

Plusieurs algorithme par opération





Plusieurs plan possibles ➡ **Évaluation des coûts pour trouver le meilleur plan physique**

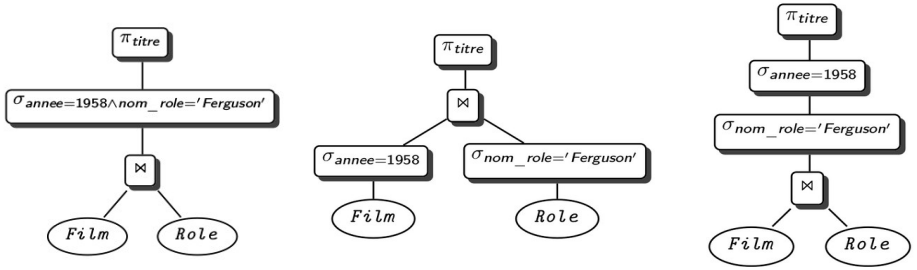


Le rôle de l'optimiseur

Trouver les expressions équivalentes

Requête SQL ➡ **Plan d'exécution logique - PEL (l'algèbre)**

```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```

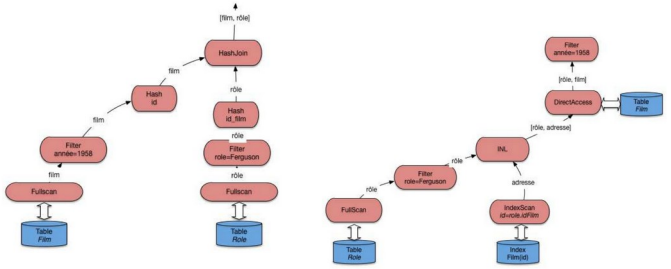


Choisir le bon algorithme pour chaque opération

Plan d'exécution physique – PEP (opérateurs)

Un opérateur = une opération

Plusieurs algorithme par opération

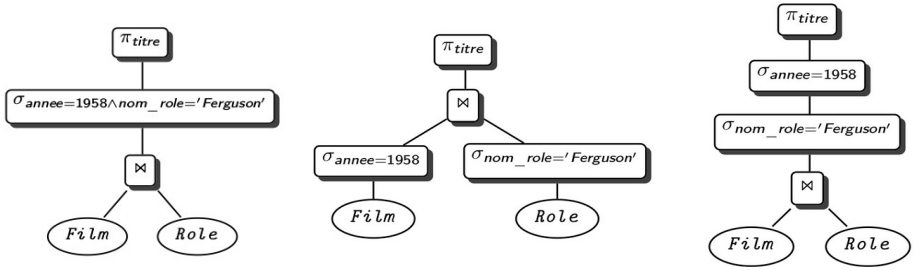


Le rôle de l'optimiseur

Trouver les expressions équivalentes

Requête SQL ➡ **Plan d'exécution logique - PEL (l'algèbre)**

```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```

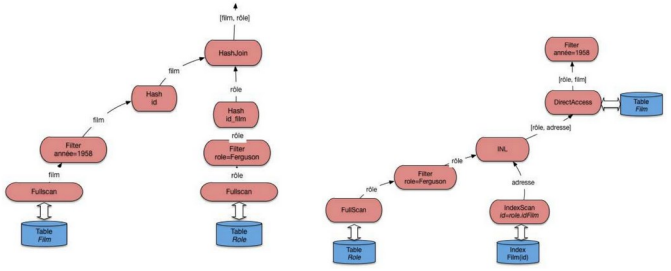


Choisir le bon algorithme pour chaque opération

Plan d'exécution physique – PEP (opérateurs)

Un opérateur = une opération

Plusieurs algorithme par opération



Le rôle de l’optimiseur : la réécriture algébrique (PEL)

- **Problème :**
  - suivant l'ordre des opérateurs algébriques dans un arbre, le coût d'exécution est diffèrent

- **Pourquoi?**
  - le coût des opérateurs varient en fonction du volume des données traitées : plus le nombre de n-uplets des relations traitées est petit, plus les coûts cpu et d'E/S sont minimisés
  - certains opérateurs diminuent le volume des données (restriction, projection, …)

➡ **Restructuration algébrique**

Le rôle de l'optimiseur : la réécriture algébrique (PEL)

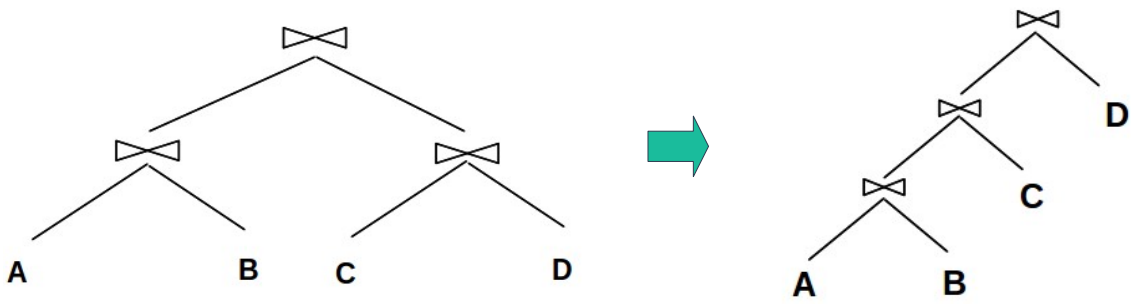
- **Trouver les expressions équivalentes**
  - l'algèbre permet d'obtenir une version opératoire de la requête
  - les équivalences algébriques permettent d'explorer un ensemble de plans
  - l'optimiseur évalue le coût (entrée / sortie) de chaque plan : différentes fonctions / modèles de coût existantes

Exemples de règles de réécriture

1. **Commutativité des jointures** : $R \bowtie S \equiv S \bowtie R$
2. **Regroupement des sélections** : $\sigma_{A='a' \wedge B='b'}(R) \equiv \sigma_{A='a'}(\sigma_{B='b'}(R))$
3. **Commutativité de σ et de π** : $\pi_{A_1,A_2,\ldots A_p}(\sigma_{A_i='a'}(R)) \equiv \sigma_{A_i='a'}(\pi_{A_1,A_2,\ldots A_p}(R))$
4. **Commutativité de σ et de $\bowtie$** : $\sigma_{A='a'}(R[\ldots A \ldots] \bowtie S) \equiv \sigma_{A='a'}(R) \bowtie S$

Le rôle de l'optimiseur : la réécriture algébrique (PEL)

- **Trouver les expressions équivalentes**
  - MAIS impossible d'énumérer tous les plans possibles
    - ➡ Utilisations d'heuristiques
  - Heuristique classique = réduction de la taille des données
    - ➡ Opérations réductrices (sélections et projections) groupées sur chaque relation le plus tôt possible, et jointures regroupées
      - grouper les restrictions aux feuilles (dégrouper puis descendre)
      - descendre les projections
      - regrouper les jointures du même coté de l'arbre (deep-left plan)



Un exemple de réécriture algébrique (PEL)

- **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 ;

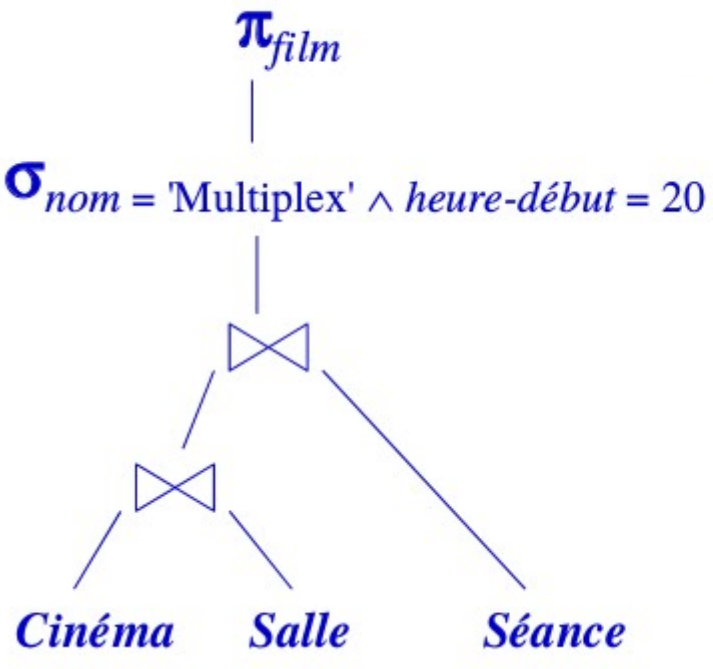
- **Expression algébrique**

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

Un exemple de réécriture algébrique (PEL)

- **Arbre algébrique de requête**

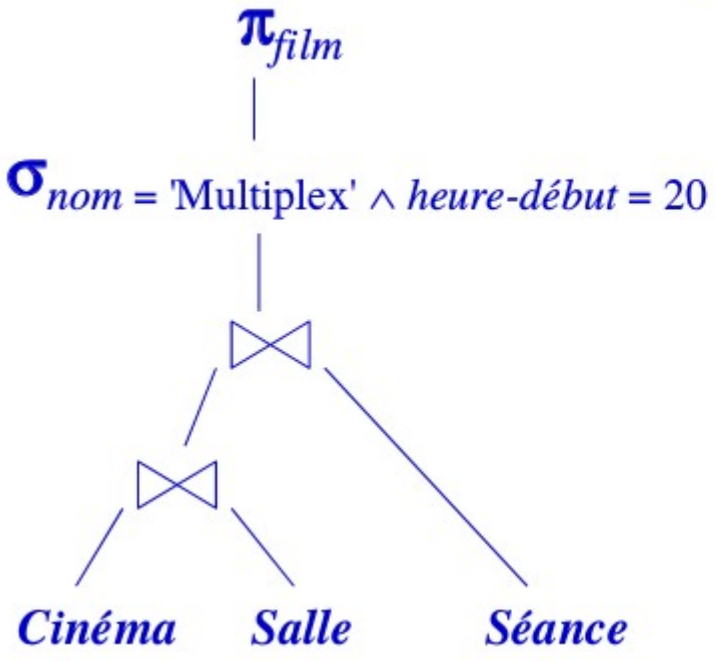
$\pi_{film}(\sigma_{nom = \text{'Multiplex'} \wedge heure\text{-}début=20}((\text{Cinéma} \bowtie \text{Salle}) \bowtie \text{Séance}))$



Un exemple de réécriture algébrique (PEL)

- **Arbre algébrique de requête**

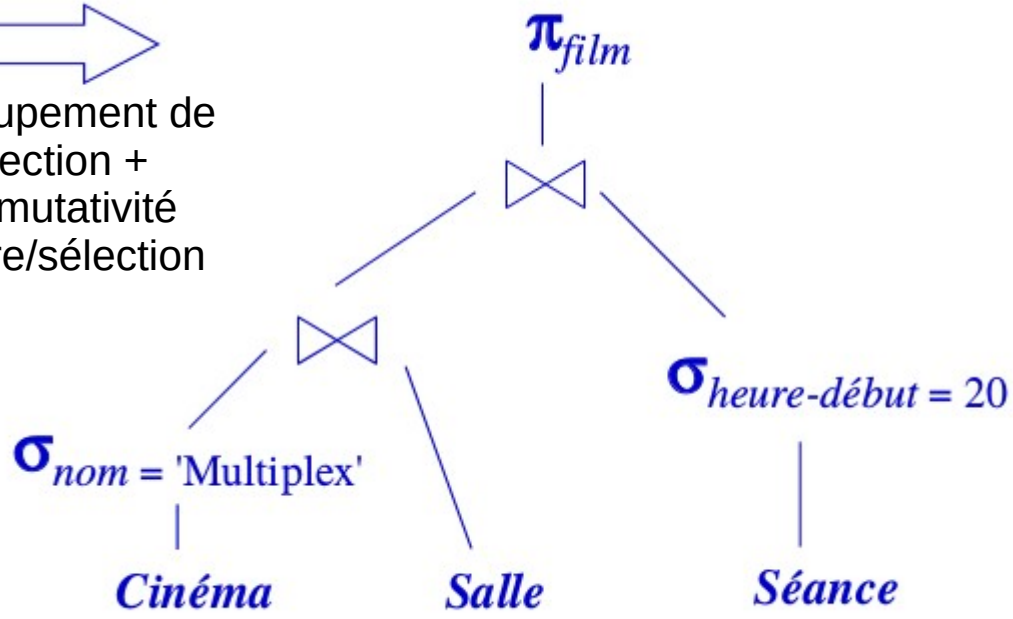
$\pi_{film}(\sigma_{nom = 'Multiplex' \wedge heure\text{-}début=20}((Cinéma \bowtie Salle) \bowtie Séance))$



Règles de réécriture utilisées :

⇒

Regroupement de sélection + commutativité jointure/sélection

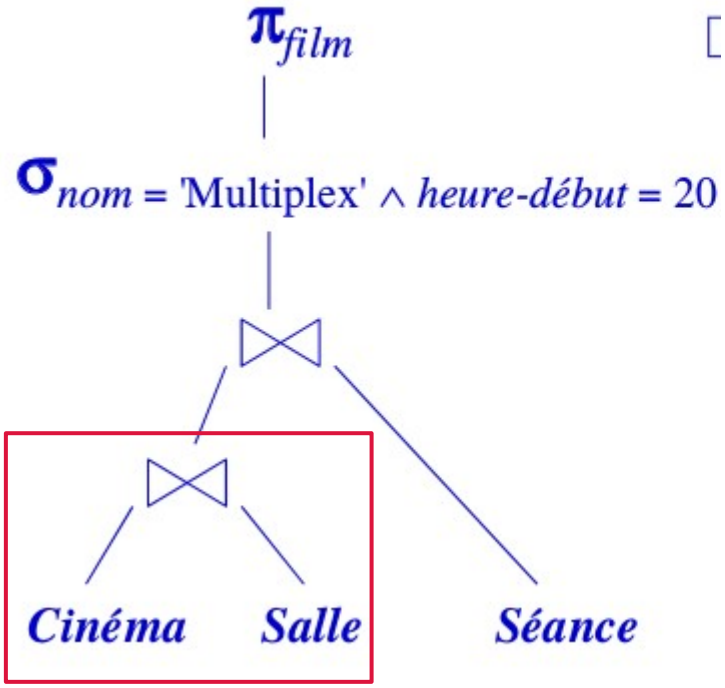


$\pi_{film}(\sigma_{nom = 'Multiplex' \wedge heure\text{-}début=20} Séance) \bowtie ((\sigma_{nom = 'Multiplex'} Cinéma) \bowtie Salle)))$

Un exemple de calcul de coût (PEL)

- **Hypothèses (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

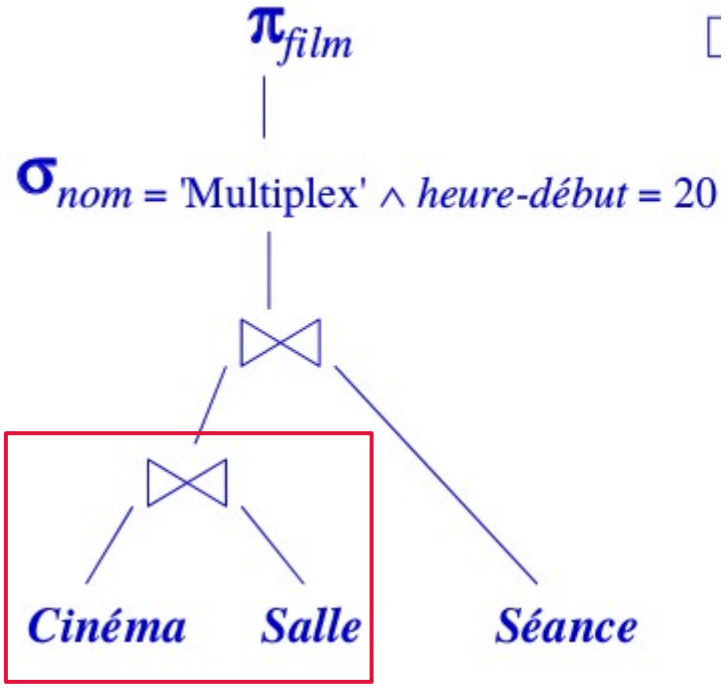
- **Plan 1 :**
  - Jointure : on lit 4 * 6 = 24 lignes
    et on produit 50 % * 6 = 3 lignes



Un exemple de calcul de coût (PEL)

- **Hypothèses (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

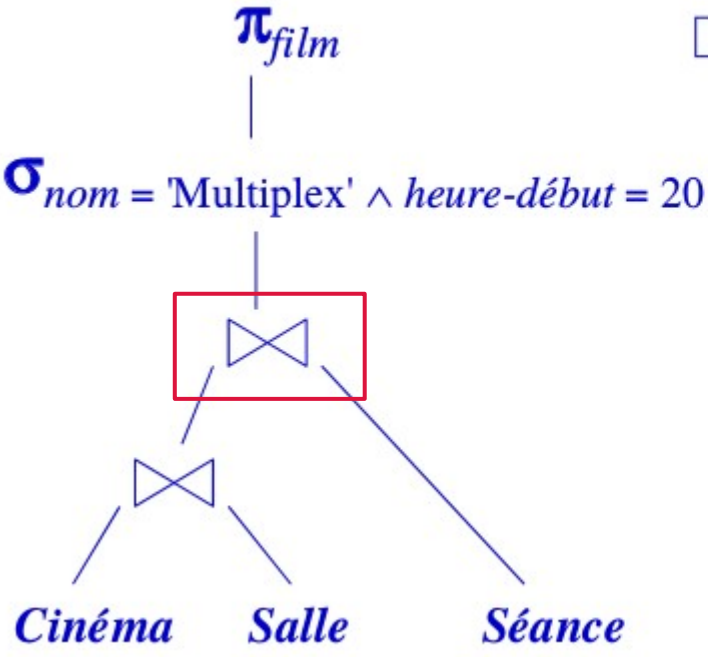
- **Plan 1 :**
  - Jointure : on lit 4 * 6 = 24 lignes
    et on produit 50 % * 6 = 3 lignes
    => Sélectivité de la jointure = 0,5
    (la moitié des salles sont des salles de Cinéma)



Un exemple de calcul de coût (PEL)

- **Hypothèses (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

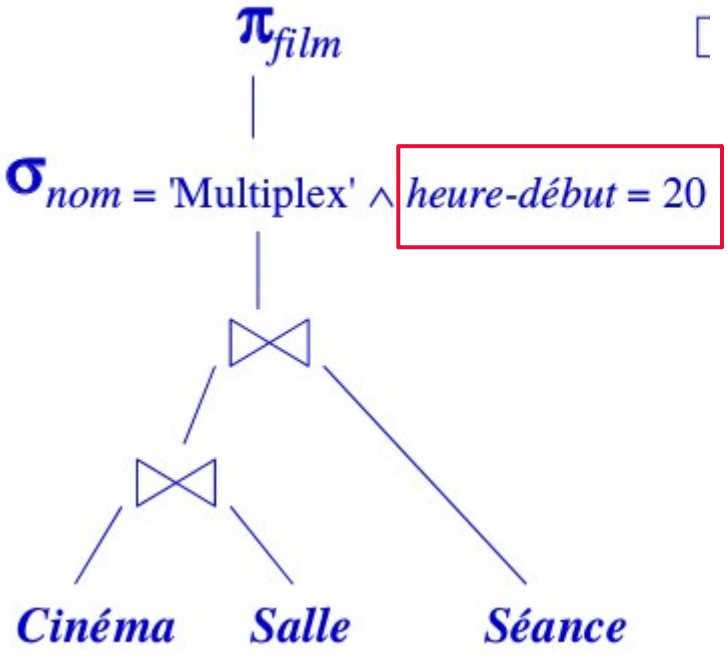
- **Plan 1 :**
  - Jointure : on lit 4 * 6 = 24 lignes
    et on produit 50 % * 6 = 3 lignes
  - Jointure : on lit 3 * 50 = 150 lignes
    et on produit 50 lignes
    => Sélectivité de la jointure = 1
    (toutes les séances sont des séances de salles de cinéma)



Un exemple de calcul de coût (PEL)

- **Hypothèses (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

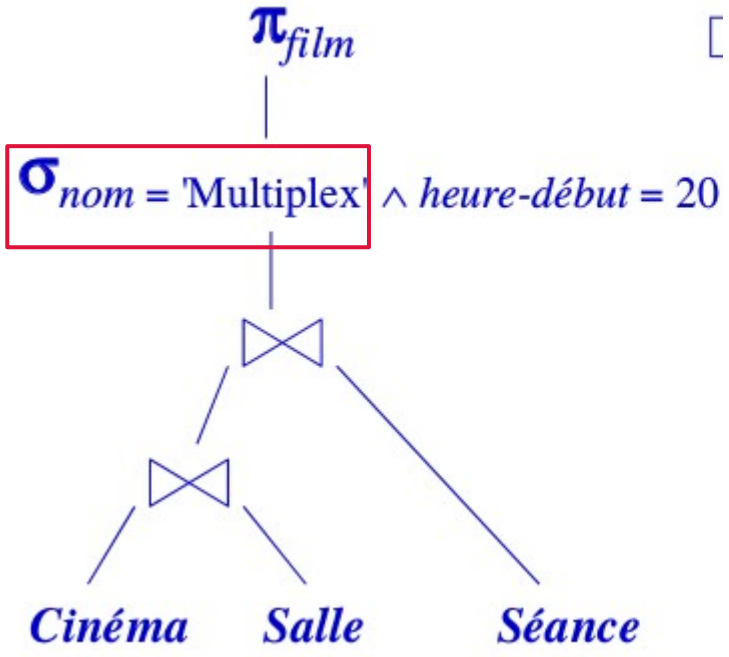
- **Plan 1 :**
  - Jointure : on lit 4 * 6 = 24 lignes
    et on produit 50 % * 6 = 3 lignes
  - Jointure : on lit 3 * 50 = 150 lignes
    et on produit 50 lignes
  - Sélection : on lit 50 lignes
    et on produit 50 % * 50 = 25 lignes
    => Sélectivité de la restriction = 0,5
    (la moitié des séances sont aprés 20h)



Un exemple de calcul de coût (PEL)

- **Hypothèses (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

- **Plan 1 :**
  - Jointure : on lit 4 * 6 = 24 lignes
    et on produit 50 % * 6 = 3 lignes
  - Jointure : on lit 3 * 50 = 150 lignes
    et on produit 50 lignes
  - Sélection : on lit 50 lignes
    et on produit 50 % * 50 = 25 lignes
  - Sélection : on lit 25 lignes
    et on produit 20 % * 25 = 5 lignes
    => Sélectivité de la restriction = 0,2
    (20 % des cinémas sont des Multiplex)

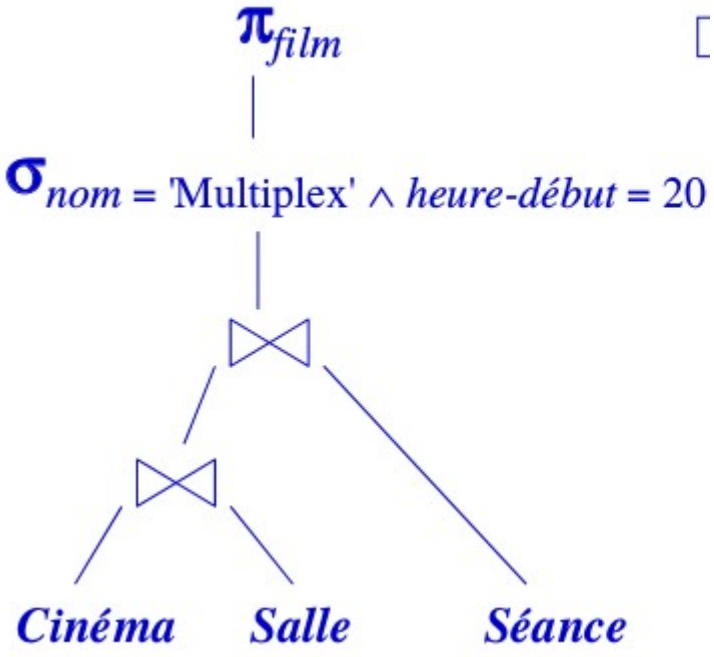


Un exemple de calcul de coût (PEL)

- **Hypothèses (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

- **Plan 1 :**
  - Jointure : on lit 4 * 6 = 24 lignes
    et on produit 50 % * 6 = 3 lignes
  - Jointure : on lit 3 * 50 = 150 lignes
    et on produit 50 lignes
  - Sélection : on lit 50 lignes
    et on produit 50 % * 50 = 25 lignes
  - Sélection : on lit 25 lignes
    et on produit 20 % * 25 = 5 lignes
  - On laisse de côté la projection (même coût dans les deux cas et même nombre de lignes)

➡ Coût (E/S) : 24E + 3S + 150E + 50S +50E + 25S + 25E + 5S = 332 lignes E/S

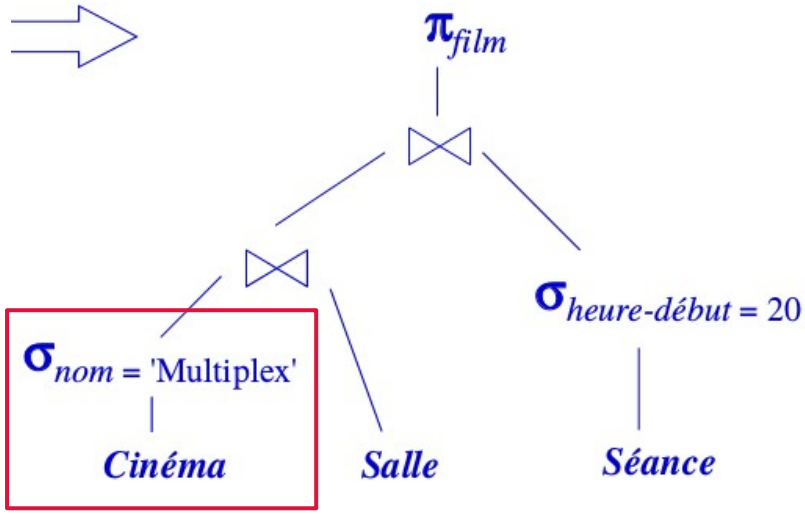


Un exemple de calcul de coût (PEL)

- **Hypothèse (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

- **Plan 1 :** coût (E/S) = 332 lignes E/S

- **Plan 2 :**
  - Sélection : on lit 4 lignes
    et on produit 20 % * 4 = 1 lignes
    => Sélectivité de la restriction = 0,2
    (20 % des cinémas sont des Multiplex)



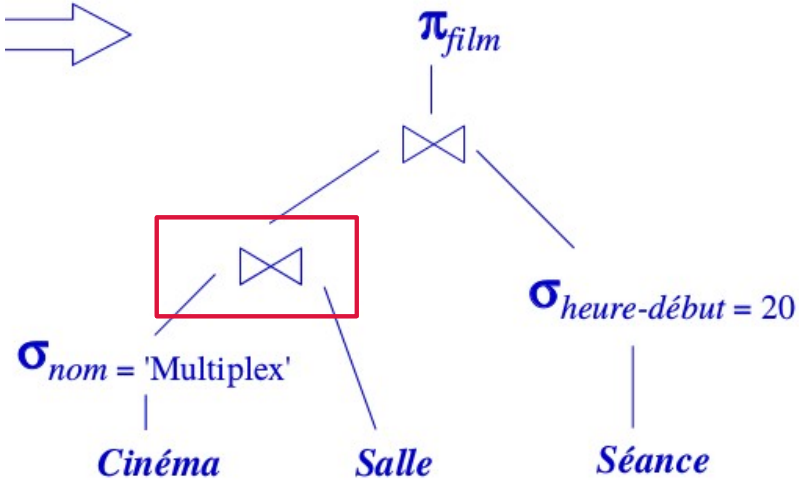
Plan 2 optimal

Un exemple de calcul de coût (PEL)

- **Hypothèse (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

- **Plan 1 :** coût (E/S) = 332 lignes E/S

- **Plan 2 :**
  - Sélection : on lit 4 lignes
    et on produit 20 % * 4 = 1 lignes
  - Jointure : on lit 1 * 6 = 6 lignes
    et on produit 50 % * 6 = 3 lignes
    => Sélectivité de la jointure = 0,5
    => Nombre de lignes MAX : dans le pire des cas, toutes les salles sont des salles du cinéma ‘Mutiplex’



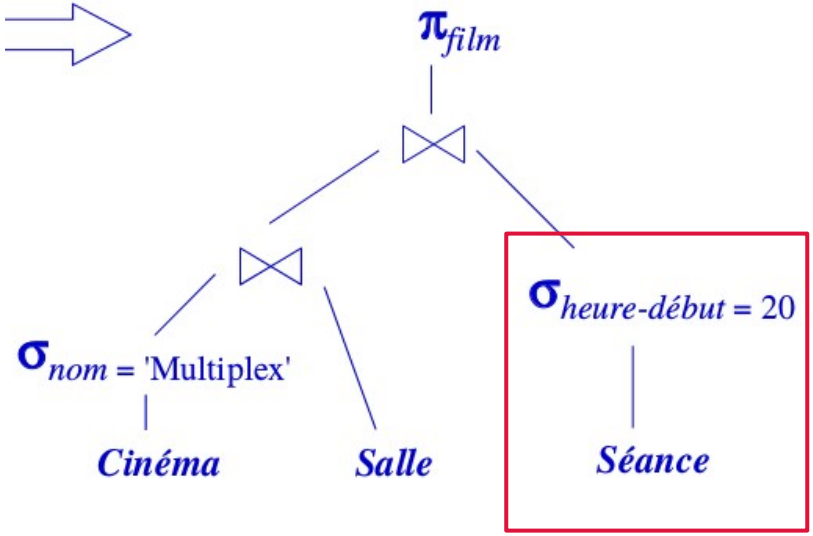
Plan 2 optimal

Un exemple de calcul de coût (PEL)

- **Hypothèse (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

- **Plan 1 :** coût (E/S) = 332 lignes E/S

- **Plan 2 :**
  - Sélection : on lit 4 lignes
    et on produit 20 % * 4 = 1 lignes
  - Jointure : on lit 1 * 6 = 6 lignes
    et on produit 50 % * 6 = 3 lignes
  - Sélection : on lit 50 lignes
    et on produit 50 % * 50 = 25 lignes
    => Sélectivité de la restriction = 0,5
    (la moitié des séances sont aprés 20h)



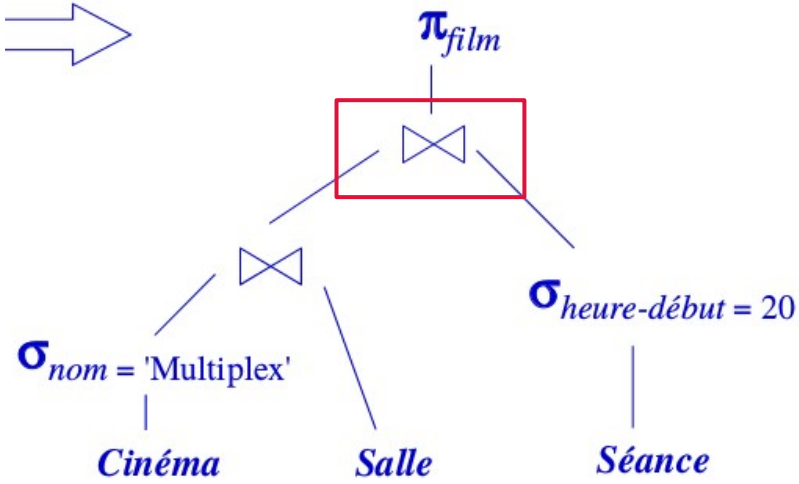
Plan 2 optimal

Un exemple de calcul de coût (PEL)

- **Hypothèse (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

- **Plan 1 :** coût (E/S) = 332 lignes E/S

- **Plan 2 :**
  - Sélection : on lit 4 lignes
    et on produit 20 % * 4 = 1 lignes
  - Jointure : on lit 1 * 6 = 6 lignes
    et on produit 50 % * 6 = 3 lignes
  - Sélection : on lit 50 lignes
    et on produit 50 % * 50 = 25 lignes
  - Jointure : on lit 25 * 3 = 75 lignes
    et on produit 25 lignes
    => Sélectivité de la jointure = 1
    => Nombre de lignes MAX



Plan 2 optimal

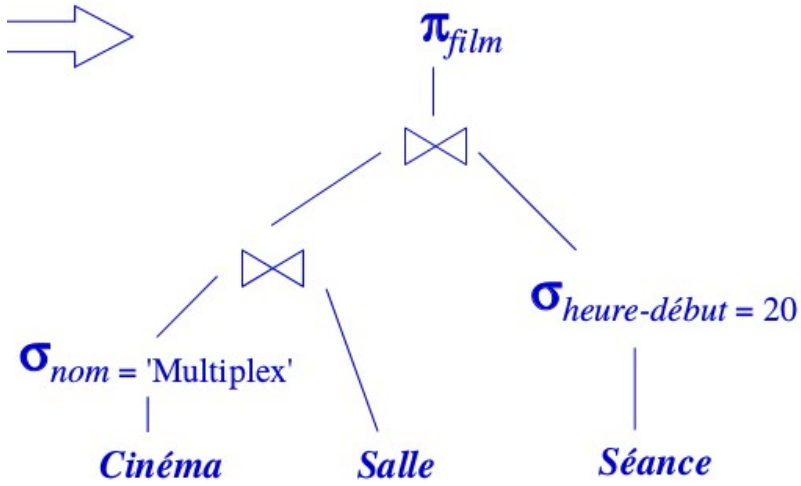
Un exemple de calcul de coût (PEL)

- **Hypothèse (en nombre de lignes) :**
  - Cinéma : 4 lignes dont 20 % de Multiplex
  - Salle : 6 lignes dont 50 % des salles de Cinéma
  - Séance : 50 lignes et 50 % des séances après 20h

- **Plan 1 :** coût (E/S) = 332 lignes E/S

- **Plan 2 :**
  - Sélection : on lit 4 lignes
    et on produit 20 % * 4 = 1 lignes
  - Jointure : on lit 1 * 6 = 6 lignes
    et on produit 50 % * 6 = 3 lignes
  - Sélection : on lit 50 lignes
    et on produit 50 % * 50 = 25 lignes
  - Jointure : on lit 25 * 3 = 75 lignes
    et on produit 25 lignes
  - On laisse de côté la projection (même coût dans les deux cas et même nombre de lignes)

➡ Coût (E/S) : 4E + 1S + 6E + 3S + 50E + 25S + 75E + 25S = 189 lignes E/S



➡ Plan 2 optimal

Un exemple de réécriture algébrique qui échoue (PEL)

- **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: $\pi_{\text{réalisateur}}(\sigma_{\text{heure-début} > 14}(\text{Film} \triangleright\!\bowtie \text{Séance}))$
  - Optimisée: $\pi_{\text{réalisateur}}(\text{Film} \triangleright\!\bowtie \sigma_{\text{heure-début} > 14}(\text{Séance}))$
- **Hypothèses**
  - Film occupe 8 lignes
  - Séance occupe 50 lignes, 90% des séances sont après 14h et 20 % des séances concernent des films

Un exemple de réécriture algébrique qui échoue (PEL)

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

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

- **Plan optimisé: $\pi_{\text{réalisateur}}(\text{Film} \bowtie \sigma_{\text{heure-début} > 14}(\text{Séance}))$**
  - Sélection: on lit 50 lignes et on produit 90% * 50 = 45 lignes de séances
  - Jointure: on lit 8 * 45 = 360 lignes et on produit 20% * 45 = 9 lignes

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

➡ **D'aprés la fonction de coût utilisée, le meilleur plan est le plan initial !**
Cas rare: ici la jointure est plus sélective que la sélection

Détails sur le calcul du coût : un exemple de fonction de coût

Fonction / Modèle de coût

- **Facteur de sélectivité S**
  - Proportion de n-uplets du produit cartésien des relations touchées qui satisfont une condition

- **Exemple :**

  SELECT * FROM R1, R2

  ➡ S = 1

  SELECT * FROM R1 WHERE A = valeur

  ➡ S = 1 / CARD(A) avec un modèle uniforme

Détails sur le calcul du coût : un exemple de fonction de coût

Sélectivité des Restrictions

TAILLE (σ(R)) = s * TAILLE(R) avec :

- s(A = valeur) = 1 / CARD(A)
- s(A > valeur) = (max(A) - valeur) / (max(A) - min(A))
- s(A < valeur) = (valeur - min(A)) / (max(A) - min(A))
- s(A IN liste valeurs) =
  (1/CARD(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)

Détails sur le calcul du coût : un exemple de fonction de coût

- **TAILLE(R1 $\bowtie_{A=B}$ R2) = p * TAILLE(R1) * TAILLE(R2)**
  - p dépend du type de jointure et de la corrélation des colonnes :
    - p = 0 si aucun n-uplet n'est joint
    - p = 1 / MAX(CARD(A),CARD(B)) si distribution uniforme équiprobable des attributs A et B sur un même domaine (col. Join.)
    - p = 1 si produit cartésien

- **Cas particulier :**
  - Si A est clé de R1 et B est clé étrangère de R2 alors **TAILLE(R1 $\bowtie_{A=B}$ R2) = TAILLE(R2)**

Conclusion sur la réécriture algébrique (PEL)

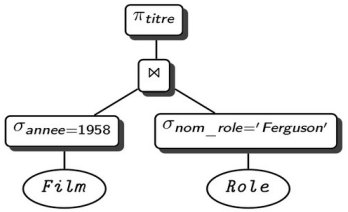
- **La réécriture algébrique est nécessaire, mais pas suffisante**
- **Il faut tenir compte d'autres critères:**
  - **Les chemins d'accès aux données (selon l'organisation physique)**
    - On peut accéder aux données d'une table par accès séquentiel, par index, par hachage, etc.
  - **Les différents 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.

Le rôle de l'optimiseur

Trouver les expressions équivalentes

Requête SQL ➡ **Plan d'exécution logique - PEL (l'algèbre)**

```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```

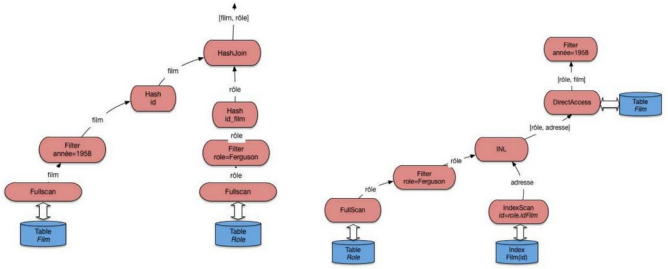


Choisir le bon algorithme pour chaque opération

Plan d'exécution physique - PEP (opérateurs)

Un opérateur = une opération

Plusieurs algorithme par opération



Le rôle de l’optimiseur : les opérateurs (PEP)

- **Les opérations de l’algèbre relationnelle peuvent être évaluée à l’aide de plusieurs algorithmes**

  ➡ une même expression d’algèbre relationnelle peut être évaluée de plusieurs façons différentes

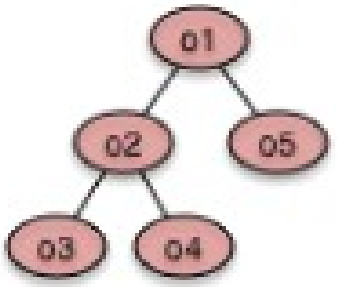
- **Une expression annotée indiquant les méthodes utilisées est un plan d’exécution physique**
  - Par exemple, s’il existe, il est possible d’utiliser un index pour obtenir tous les employés dont le salaire est supérieur à 30000
  - Ou toute la table peut être lue et seuls les employés dont le salaire est supérieur à 30000 sont conservés

Le rôle de l’optimiseur : les opérateurs (PEP)

- **Tout opérateur est implanté sous forme d’un itérateur**

- **Trois fonctions :**
  - open : initialise les ressources et positionne le curseur
  - next : ramène l’enregistrement courant et se place sur l’enregistrement suivant
  - close : libère les ressources

- **Un plan d'exécution est un arbre d'itérateurs**
  - un itérateur consomme des nuplets d’autres itérateurs source ou de données
  - un itérateur produit des nuplets à la demande



Plan exécution *physique*

Le rôle de l’optimiseur : les opérateurs (PEP)

- **Rôle des itérateurs : principes essentiels**
  - production à la demande : le serveur n'envoie un enregistrement au client que quand ce dernier le demande
  - Pipeline : on essaie d'éviter le stockage en mémoire de résultats intermédiaires (le résultat est calculé au fur et à mesure

  ➡ évite d'avoir à stocker des résultats intermédiaire

  ➡ temps de réponse minimisé mais attention aux opérateurs bloquants

  - temps de réponse : temps pour obtenir le premier nuplet
  - temps d’exécution : temps pour obtenir tous les nuplets.

- **Opérateur bloquant**

  ➡ on additionne le temps d’exécution et le temps de d’exécution

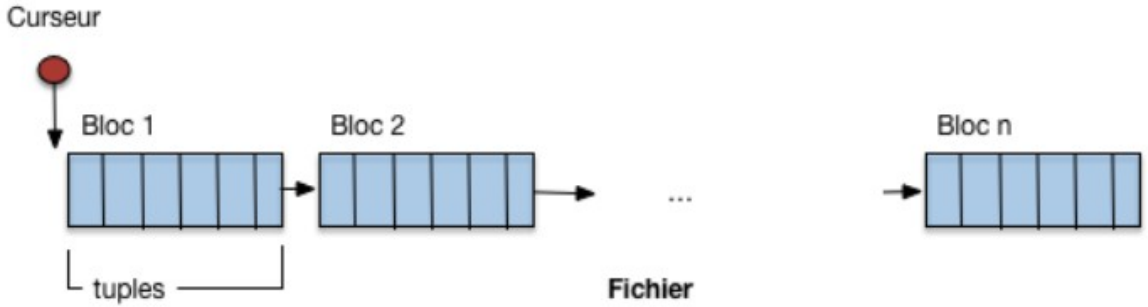
```
select min(date) from T
```

Le rôle de l’optimiseur : les opérateurs (PEP)

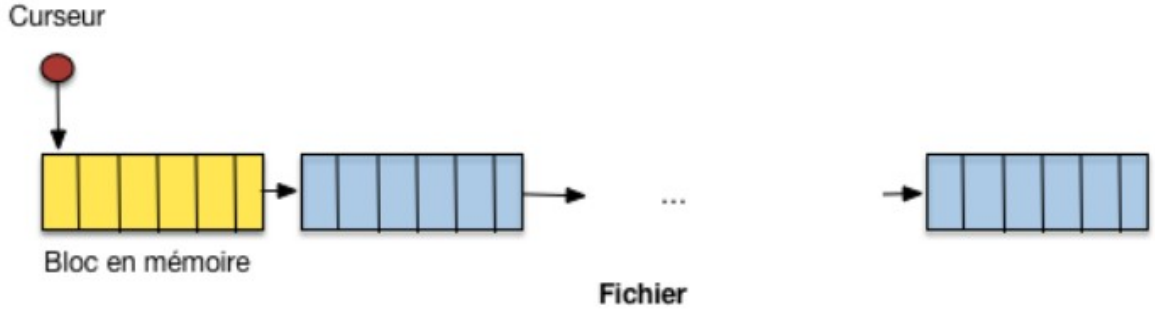
- **3 principaux types d’opérateurs et plusieurs algorithmes pour chaque opération:**
  - Accès aux données (via les tables et les index)
    - parcours séquentiel de la table (*FullScan*)
    - parcours d’index (*IndexScan*)
    - accès par adresse (*DirectAccess*)
    - test de la condition (*Filter*)
  - Jointures (sans / avec index)
    - jointure par boucles imbriquées (*Nested loop join*) indexée ou non indexée
    - jointure par tri-fusion (*Merge sort join*)
    - jointure par hachage (*Hash join*)
  - Tri et regroupement
    - tri externe (même type d’algorithme que pour les jointures)

Le rôle de l'optimiseur : l'opérateur d'accès aux données *FullScan (PEP)*

- **Curseur positionné avant le premier nuplet**
- **open() = phase d'initialisation de l'opérateur**

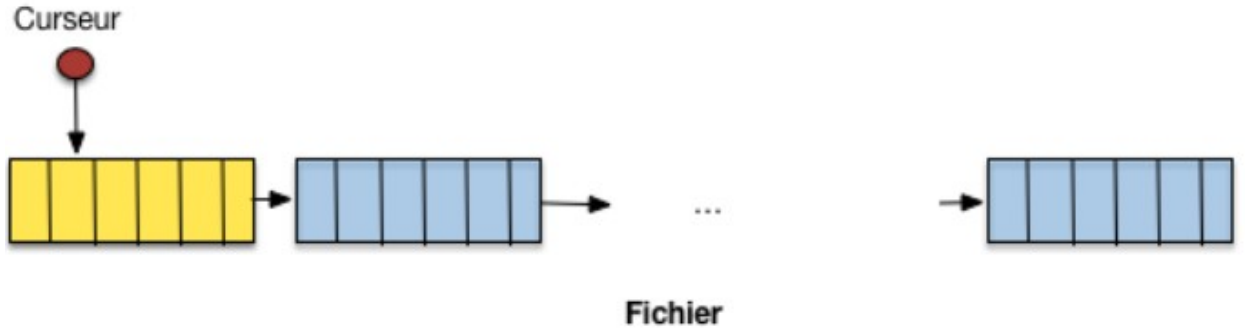


- **1er next() = accès au premier bloc, placé en mémoire**
  - Le curseur se place sur le premier nuplet, qui est retourné comme résultat. Le temps de réponse est minimal.

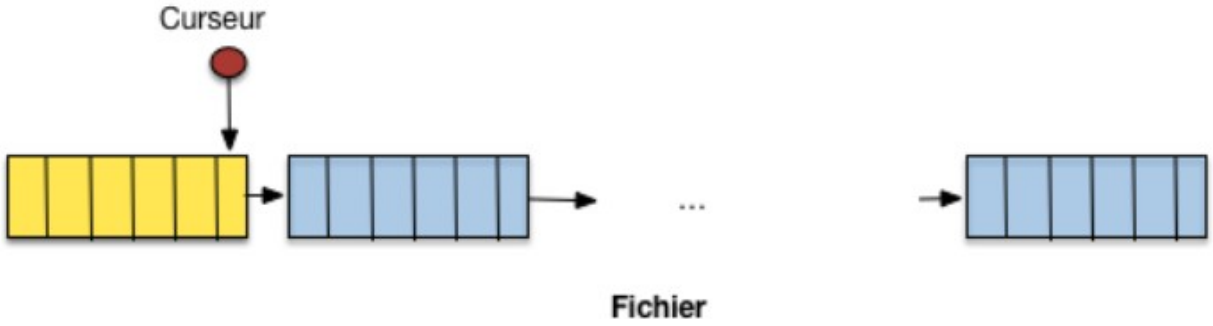


Le rôle de l'optimiseur : l'opérateur d'accès aux données *FullScan (PEP)*

- **2ème next() = avancée d'un cran dans le parcours du bloc**



- **Après plusieurs next() : curseur positionné sur le dernier nuplet du bloc**

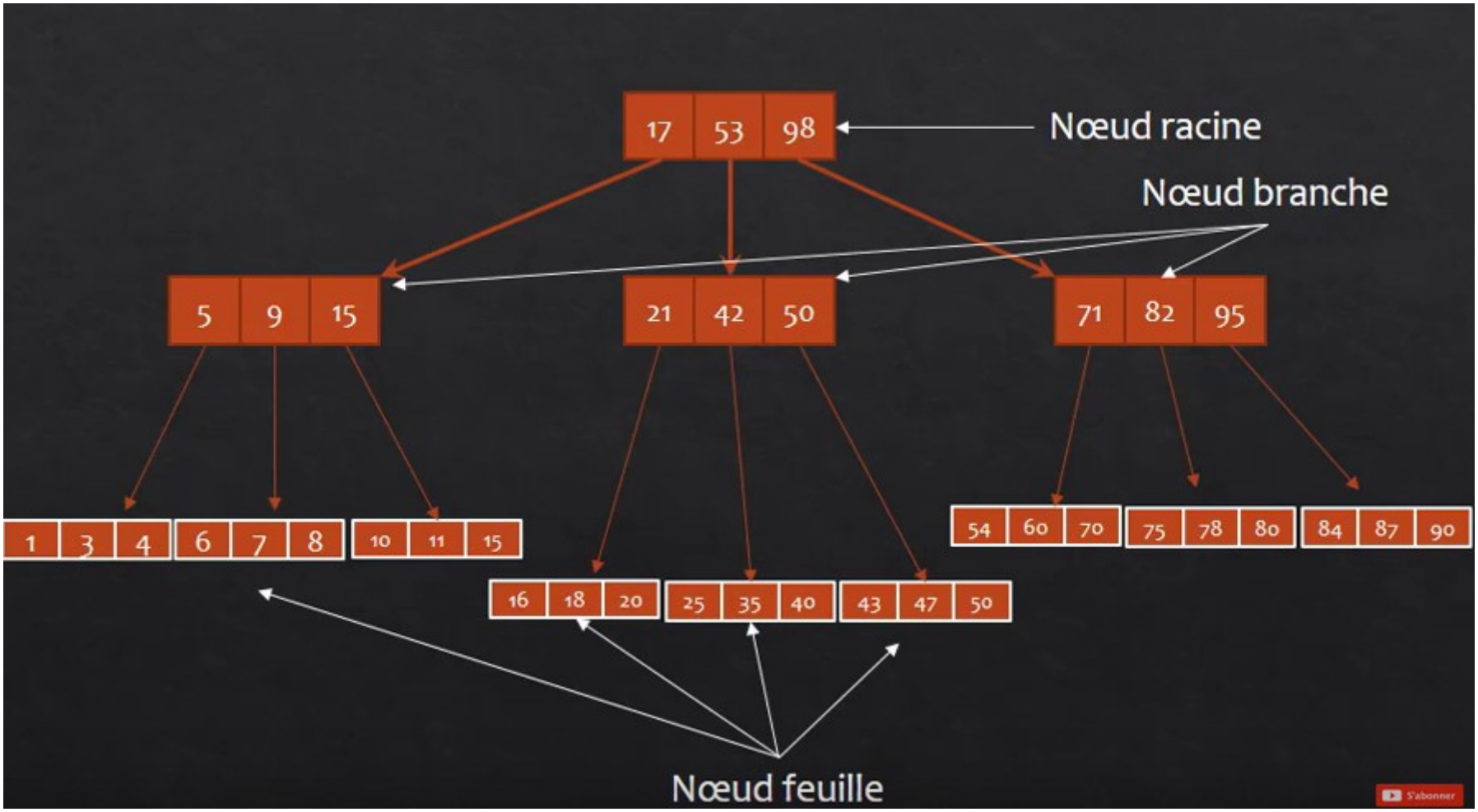


- **Appel suivant à next() = charge du second bloc en mémoire**

➡ **Besoin en mémoire réduit (1 bloc) ; temps de réponse très court.**

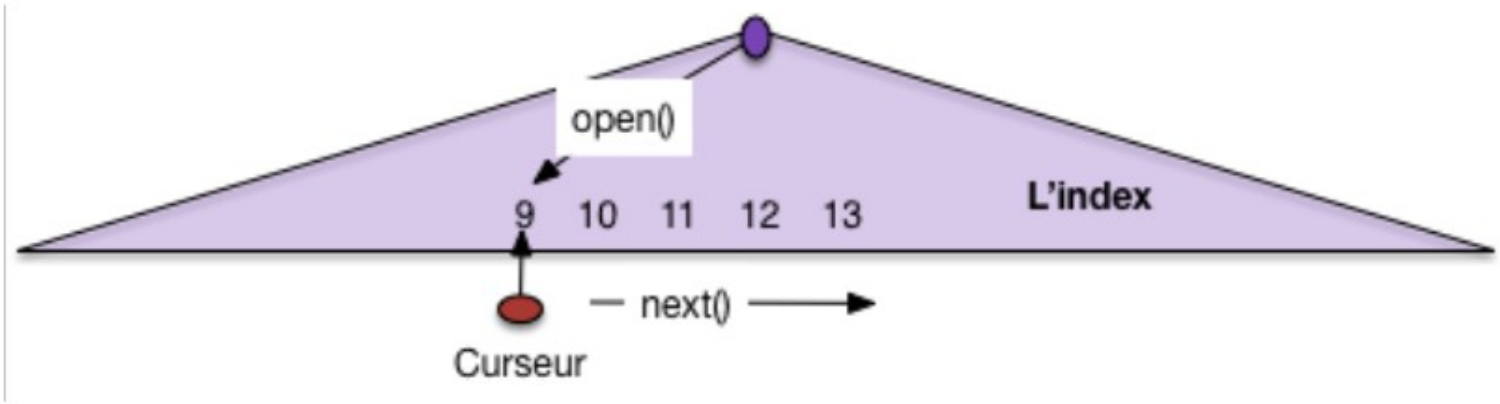
Le rôle de l'optimiseur : l'opérateur d'accès aux données *IndexScan (PEP)*

- **Index = arbre équilibré (B tree)**



Le rôle de l'optimiseur : l'opérateur d'accès aux données *IndexScan (PEP)*

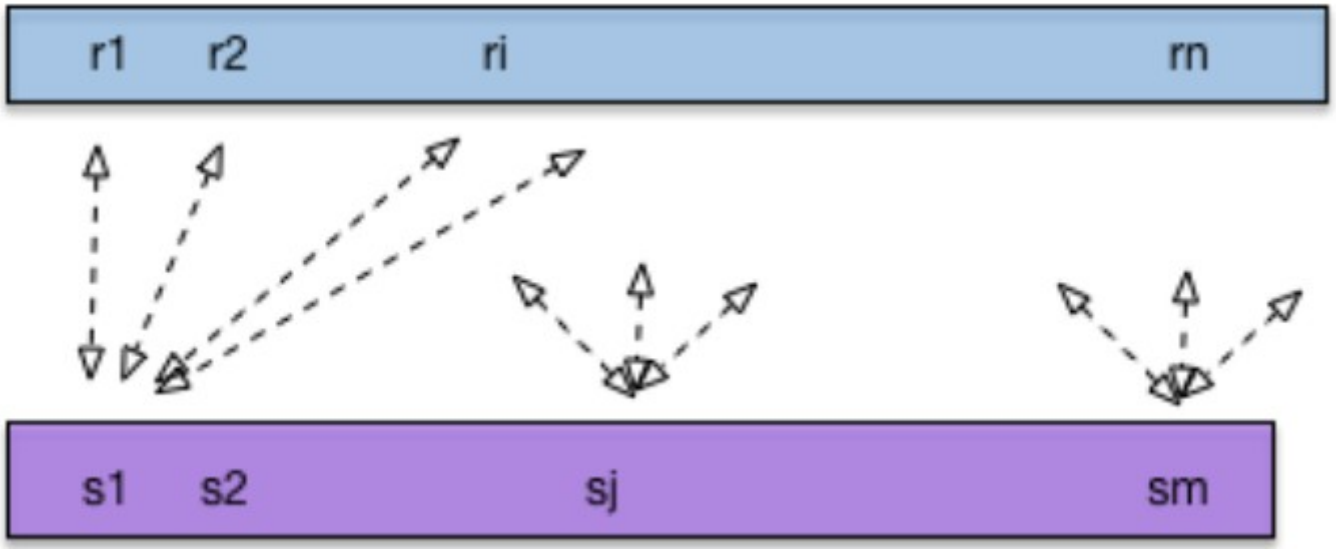
- **Index = arbre équilibré (B tree)**
- **Pendant le open() : parcours de la racine vers la feuille**
- **À chaque appel à next() : parcours en séquence des feuilles**



➡ **Très efficace, quelques lectures logiques (index en mémoire)**

Le rôle de l'optimiseur : l'opérateur de jointure *Nested loop join (PEP)*

- **Énumérer toutes les solutions possibles : la table à droite de la jointure est confrontée à chaque tuple de la table à gauche**



- **Si la table à droite est indexée sur l'attribut qui sert pour la jointure, cette approche peut donner des résultats corrects (sinon elle peut s'avérer coûteuse)**

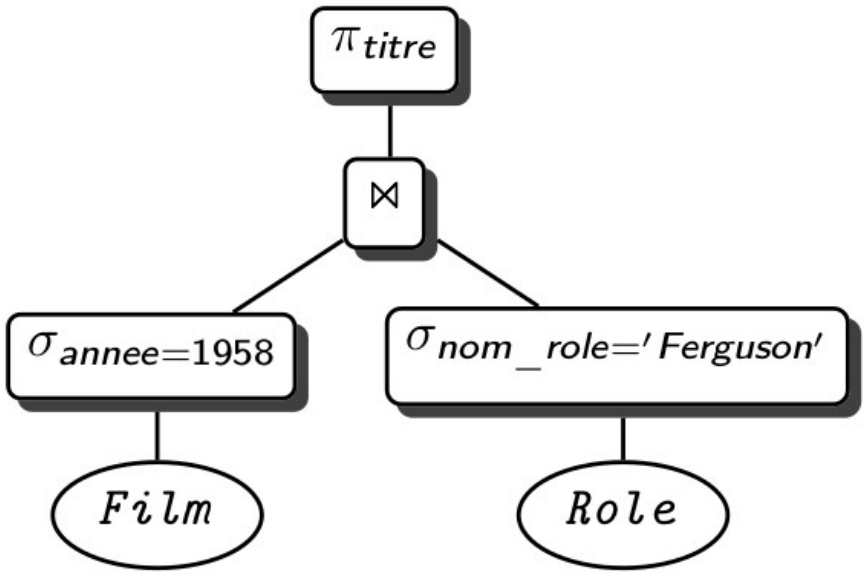
Le rôle de l’optimiseur : l’opérateur de jointure *Hash Join (PEP)*

- **Le plus efficace dans le meilleur des cas**
  - Très rapide quand une des deux tables est petite (n fois la taille de la mémoire avec n petit, <3)
  - Pas très robuste (efficacité dépend de plusieurs facteurs : fonction de hachage, nb/taille casiers, ...)

- **Idée de base**
  - Les données des deux tables sont lues en mémoire, et les données de la table principale sont hachées.
  - Pour chaque ligne de la table secondaire, la fonction de hachage est appliquée sur les colonnes de la jointure pour décider s'il y a un match ou non avec une ligne de la table principale.

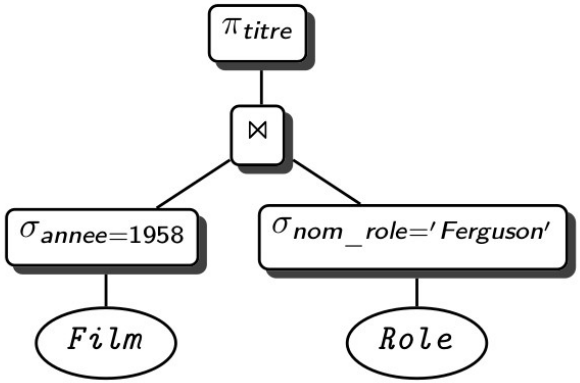
Un exemple de plan d'exécution physique (PEP)

```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```



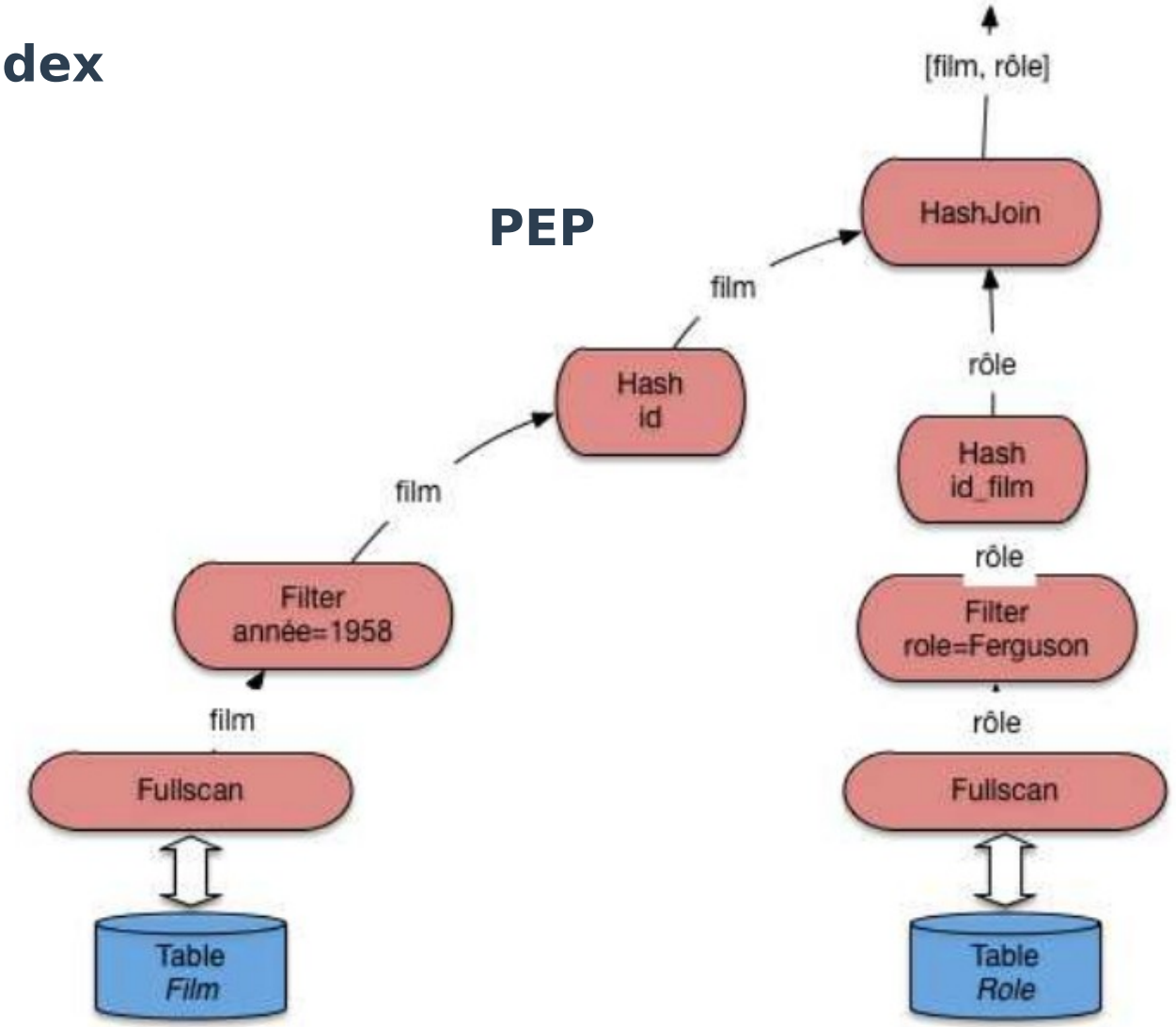
Un exemple de plan d'exécution physique (PEP)

- **Sans utiliser d'index**



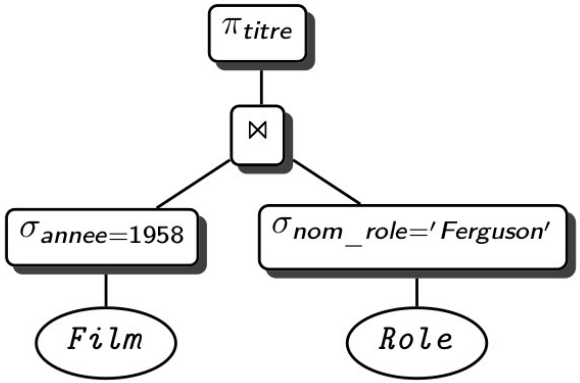
PEL

PEP



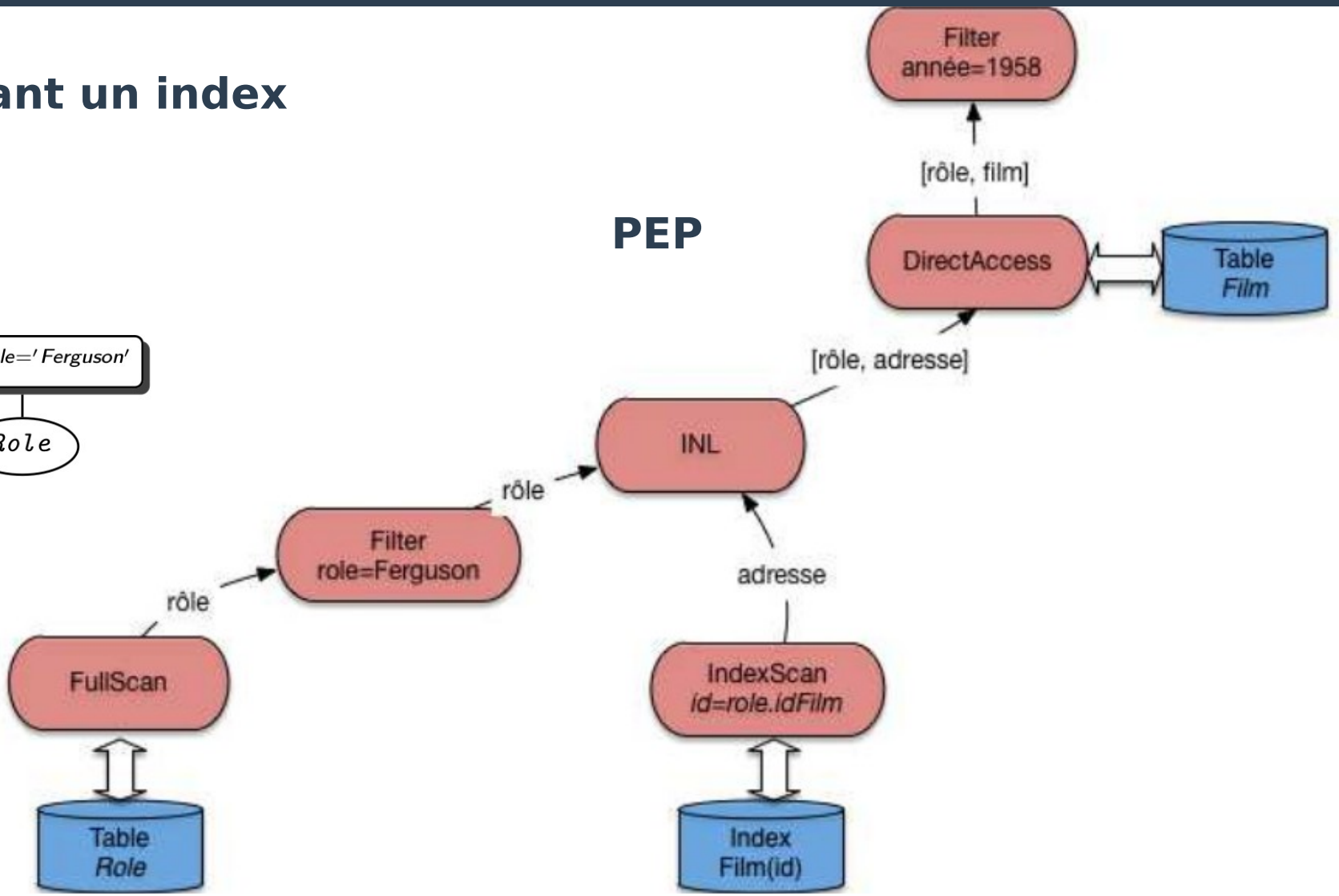
Un exemple de plan d'exécution physique (PEP)

- **En utilisant un index**



PEL

PEP



Le rôle de l’optimiseur : les opérateurs (PEP)

- **Le coût correspond au temps nécessaire pour obtenir le résultat d’une requête : plusieurs facteurs (accès disques, CPU, échange réseau)**
- **Le coût dépend de l'algorithme (index, hachage ou balayage).**
- **Parmi tous les plans d’exécutions, celui avec le coût le moins élevé est choisi**
  - Le coût est estimé en fonction de statistiques sur les tables comme le nombre de n-uplet, la taille des n-uplets, la répartition des valeurs, ….
- **Les accès disques correspondent généralement à la plus grande part du coût d’une requête. Il est possible d’estimer le nombre d’accès disque pour une requête. Il faut prendre en compte :**
  - Le nombre de recherche de données * le temps moyen d’une recherche
  - Le nombre de lecture des blocs * le temps moyen d’une lecture de bloc
  - Le nombre de bloc à écrire * le temps moyen d’écriture d’un bloc
    - Le temps d’écriture est plus important que le temps de lecture
    - Après une écriture, une lecture est effectuée afin de vérifier qu’il n’y a pas d’erreur

Le rôle de l’optimiseur : les opérateurs (PEP)

- **Méthode statique ou encore optimisation basée sur des règles (RBO) – ancienne méthode (ORACLE < v7)**
  - en fonction des chemins possibles d’accès aux tables
  - en privilégiant en premier lieu les opérations les moins coûteuses

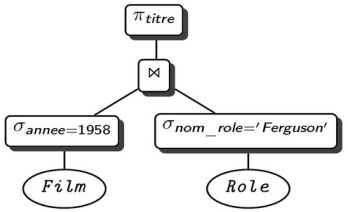
- **Méthode statistique ou encore optimisation basée sur le coût (CBO) – méthode actuelle (ORACLE ≥ v7)**
  - en fonction de l’ensemble des statistiques collectées en tâches de fond sur les différentes tables du schéma
  - 2 objectifs privilégiés : le temps d’exécution / débit (par défaut) ou le temps de réponse
  - optimisation paramétrable (mode, tuning, directive, ...)

Le rôle de l'optimiseur : résumé...

Trouver les expressions équivalentes

Requête SQL ➡ **Plan d'exécution logique - PEL (l'algèbre)**

```
select titre
from   Film f, Role r
where  nom_role ='Ferguson'
and    f.id = r.id_ilm
and    f.annee = 1958
```

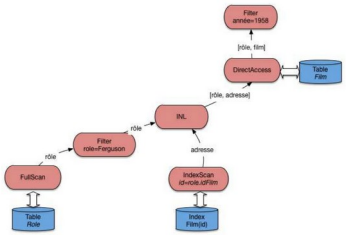


Choisir le bon algorithme pour chaque opération

Plan d'exécution physique - PEP (opérateurs)

Un opérateur = une opération

Plusieurs algorithme par opération



Le rôle de l’optimiseur : résumé…

- **Un plan d'exécution (physique) :**
  - C'est un programme combinant des opérateurs physiques (chemins d'accès et traitements de données).
- **Il a la forme d'un arbre : chaque nœud est un opérateur qui**
  - prend des données en entrée
  - applique un traitement
  - produit les données traitées en sortie
- **La phase d'optimisation proprement dite :**
  - Pour une requête, le système a le choix entre plusieurs plans d'exécution (logiques puis physiques).
  - Ils diffèrent par l'ordre des opérations, les algorithmes, les chemins d'accès.
  - Pour chaque plan on peut estimer le coût de chaque opération et la taille du résultat

➡ **Objectif : diminuer le plus vite possible la taille des données manipulées.**

L’optimiseur ORACLE

L’optimiseur ORACLE

- **L’optimiseur ORACLE suit une approche classique :**
  - Génération de plusieurs plans d’exécution.
  - Estimation du coût de chaque plan généré.
  - Choix du meilleur et exécution.
- **Tout ceci est automatique, mais il est possible d’influer, voire de forcer le plan d’exécution**
- **Paramètres pour l’estimation du coût :**
  - Les chemins d’accès disponibles.
  - Les opérations physiques de traitement des résultats intermédiaires.
  - Des statistiques sur les tables concernées (taille, sélectivité) : appel explicite à l’outil ANALYZE.
  - Les ressources disponibles.

L’optimiseur ORACLE : les paramètres

- **Principaux paramètres (doc Oracle) :**
  - OPTIMIZER_MODE
    - RULE : heuristique, utilisé seulement lorsque des statistiques ne sont pas disponibles
    - CHOOSE : Oracle choisit (RULE / ALL_ROWS) en fonction de la présence de statistiques
    - FIRST_ROW [1, 10, 100, 1000] : minimise temps de réponse (obtention des 1ère lignes)
    - ALL_ROWS : minimise le temps total
  - SORT_AREA_SIZE (taille de la zone de tri).
  - HASH_AREA_SIZE (taille de la zone de hachage).
  - HASH_JOIN_ENABLED considère les jointures par hachage.

➡ **ALTER SESSION SET [OPTIMIZER_MODE | SORT_AREA_SIZE | …] = …**

L'optimiseur ORACLE : les règles

- **Basé sur des règles**
  - Mode RULE
  - Priorité des règles

Rang	Chemin d'accès
1	Sélection par ROWID
2	Sélection d'une ligne par jointure dans une organisation par index groupant ou hachage hétérogène (CLUSTER)
3	Sélection d'une ligne par hachage sur clé candidate (PRIMARY ou UNIQUE)
4	Sélection d'une ligne par clé candidate
5	Jointure par une organisation par index groupant ou hachage hétérogène (CLUSTER)
6	Sélection par égalité sur clé de hachage (HASH CLUSTER)
7	Sélection par égalité sur clé d'index groupant (CLUSTER)
8	Sélection par égalité sur clé composée
9	Sélection par égalité sur clé simple d'index secondaire
10	Sélection par intervalle borné sur clé indexée
11	Sélection par intervalle non borné sur clé indexée
12	Tri-fusion
13	MAX ou MIN d'une colonne indexée
14	ORDER BY sur colonne indexée
15	Balayage

L'optimiseur ORACLE : les statistiques

Création des statistiques : utiliser les fonctions de DBMS_STATS

- **Calcul pour une table ou tout le shéma :**
  - `execute dbms_stats.gather_table_stats('login','nom_table');`
  - `execute dbms_stats.gather_schema_stats('login');`

- **Calcul de la taille des n-uplets et du nombre de lignes (blocs) :**

```
ANALYZE TABLE Film COMPUTE STATISTICS FOR TABLE;
```

- **Analyse des index (nombre blocs feuilles, profondeur, clustering factor (nombre de blocs concernés par la requête) :**

```
ANALYZE TABLE Film COMPUTE STATISTICS FOR ALL INDEXES;
```

- **Analyse de la distribution des valeurs (nombre valeurs distinctes, valeurs nulles, histogramme)**

```
ANALYZE TABLE Film COMPUTE STATISTICS FOR COLUMNS titre, genre;
```

L’optimiseur ORACLE : les chemins d’accès

- **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

L’optimiseur ORACLE : les opérateurs

- **ORACLE peut utiliser trois algorithmes de jointures :**
  - Boucles imbriquées quand il y a au moins un index : NESTED LOOP
  - Tri/fusion quand il n’y a pas d’index : SORT / MERGE JOIN
  - Jointure par 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
  - ...

L’optimiseur ORACLE : les directives

- **Directive (hint) utilisée pour influencer l’optimiseur**
  - imposer un opérateur spécifique,
  - faire le choix sur l’exploitation d’un index ou non, ...

➡ **Utile en mode de conception ou lorsque l’optimiseur ne choisit pas un plan optimal (ex: mauvaises statistiques)**

- **Insertion des directives dans la requête à exécuter**
  - Exemples de directives (doc ORACLE)
    - Ne pas exploiter l’index d’une table : /*+ NO_INDEX(nom_table) */
      ```
      select /*+ NO_INDEX(Commune) */ * from Commune ;
      ```
    - Utiliser l’opérateur FullScan sur une table : /*+ full(nom_table) */
      ```
      select /*+ full(f) */ * from f where nom_f like ’d%’;
      ```

- **Les directives ne peuvent être efficaces que si le traitement demandé fait partie intégrante des plans d’exécution initialement envisagés par l’optimiseur.**

L’optimiseur ORACLE : le plan d’exécution

- **Comment obtenir le plan d’exécution d’une requête ?**
  - commande pour le calculer « explain plan for requete ; »
  - Exemple : `explain plan for select * from emp where num=33000 ;`
  - commande pour l’afficher
    « Select plan_table_output from table(dbms_xplan.display()); »

```
|    0 | SELECT STATEMENT
|*   1 |    TABLE ACCESS FULL| EMP

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("NUM"=33000)
```

L’optimiseur ORACLE : le plan d’exécution

- **Comment lire un plan d’exécution ?**
  - Parcours des étapes de haut en bas jusqu’à en trouver une qui n’a pas de fille (pas d’étape indentée en dessous)
  - Traitement de cette étape sans fille ainsi que de ses sœurs (étapes de même indentation)
  - Traitement de toutes les étapes mères jusqu’à trouver une étape qui a une sœur
  - Traitement de la sœur conformément à l’étape 1.

```
-----------------------------------------------------
|   0 | SELECT STATEMENT
|   1 |  NESTED LOOPS
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP
|*  3 |    INDEX FULL SCAN           | N_DEPT_IDX |
|   4 |   TABLE ACCESS BY INDEX ROWID| DEPT       |
|*  5 |    INDEX UNIQUE SCAN         | DEPT_PK    |
-----------------------------------------------------
```

L’optimiseur ORACLE : le plan d’exécution

- **Comment lire un plan d’exécution ?**
  - Id : Identifiant de l’opérateur ;
  - Operation : type d’opération utilisée
  - Name : nom de la relation utilisée ;
  - Rows : le nombre de lignes qu’Oracle pense transférer. ,
  - Bytes : nombre d’octets qu’oracle pense transférer.
  - Cost : coût estimé par oracle

```
-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |     1 |    24 |    11  (10)| 00:00:01 |
|   1 |  SORT AGGREGATE               |               |     1 |    24 |            |          |
|*  2 |   HASH JOIN                   |               |  9940 |   232K|    11  (10)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL          | C             |  1000 |  8000 |     4   (0)| 00:00:01 |
|*  4 |    HASH JOIN                  |               |   995 | 15920 |     7  (15)| 00:00:01 |
|   5 |     TABLE ACCESS BY INDEX ROWID| B            |   100 |   800 |     2   (0)| 00:00:01 |
|*  6 |      INDEX RANGE SCAN         | B_STATUS_IDX  |   100 |       |     1   (0)| 00:00:01 |
|*  7 |     TABLE ACCESS FULL         | A             |  1000 |  8000 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

L’optimiseur ORACLE : le plan d’exécution

- **3 méthodes différentes pour appliquer les clauses where :**
  - Prédicats d'accès (« access »)
  - Prédicat de filtre d'index
  - Prédicat de filtre au niveau table
- **Information sur les prédicats :**
  - Numérotation des prédicats = colonne « Id » du plan d'exécution.
  - Étoile dans le plan d’exécution pour marquer les opérations qui ont des informations de prédicats

```
-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     1 |    24 |    11  (10)| 00:00:01 |
|   1 |  SORT AGGREGATE               |              |     1 |    24 |            |          |
|*  2 |   HASH JOIN                   |              |  9940 |   232K|    11  (10)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL          | C            |  1000 |  8000 |     4   (0)| 00:00:01 |
|*  4 |    HASH JOIN                  |              |   995 | 15920 |     7  (15)| 00:00:01 |
|   5 |     TABLE ACCESS BY INDEX ROWID| B           |   100 |   800 |     2   (0)| 00:00:01 |
|*  6 |      INDEX RANGE SCAN         | B_STATUS_IDX |   100 |       |     1   (0)| 00:00:01 |
|*  7 |     TABLE ACCESS FULL         | A            |  1000 |  8000 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("B"."ID"="C"."B_ID")
   3 - filter("C"."STATUS"='OPEN')
   4 - access("A"."STATUS"="B"."STATUS" AND "A"."B_ID"="B"."ID")
   6 - access("B"."STATUS"='OPEN')
   7 - filter("A"."STATUS"='OPEN')
```

L’optimiseur ORACLE : les statistiques des requêtes

- **Comment obtenir les statistiques des requêtes ?**
  - utilisation de l’outil Autotrace
  - permet de visualiser à la fois le plan d’exécution et les statistiques sur la requête
  - commande : « set autotrace on »
  - désactivation d’autotrace : « set autotrace off ».

```
Statistiques
----------------------------------------------------------
        196  recursive calls
          0  db block gets
         48  consistent gets
          0  physical reads
          0  redo size
       1073  bytes sent via SQL*Net to client
        396  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          5  sorts (memory)
          0  sorts (disk)
         16  rows processed
```

L’optimiseur ORACLE : les statistiques des requêtes

- **Accès mémoire : nb de pages/blocs logiques lus**
  - consistent gets : nb d’accès à une donnée consistante en RAM (non modifiée)
  - db block gets : nb d’accès à une donnée en RAM
- **Accès physiques :**
  - physical reads : nb total de pages/blocs lues sur le disque
  - recursive calls : nb d’appels à un sous-plan (requête imbriquée, tris)
- **redo size : taille du fichier de log produit (écriture, mis à jour,...)**
- **sorts (disk) : nb d’opérations de tri avec au moins une écriture**
- **sorts (memory) : nb d’opérations de tri en mémoire**

Attention : statistiques sur les requêtes différentes des statistiques sur les tables ou le schéma de la BD (cardinalité, densité, sélectivité,...)