

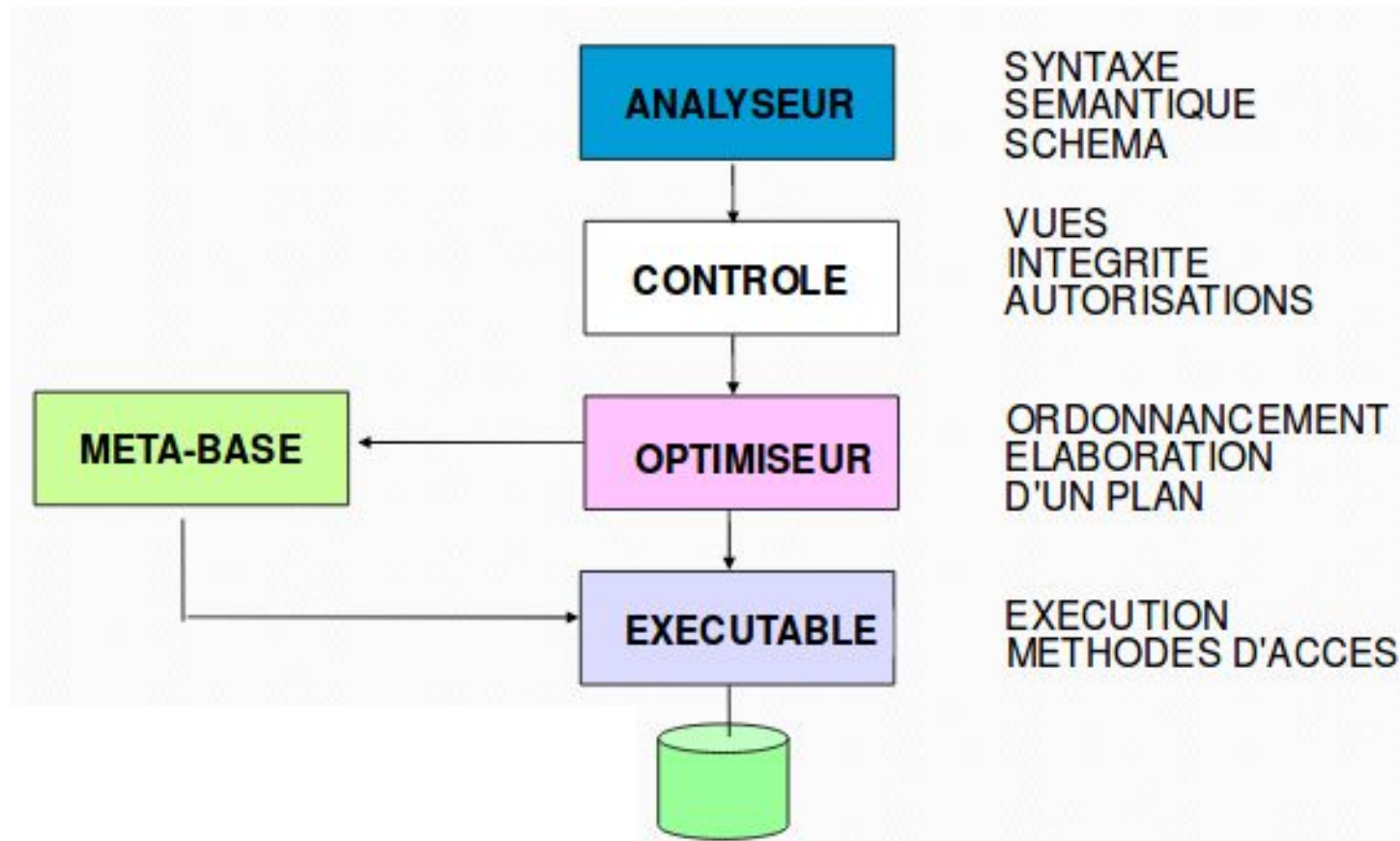
# Entrepôts de Données et Big Data - HMIN122

## 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 » ?

```
select a1, a2, ...  
from T1, T2, ...  
where ...
```

Forme  
déclarative



Forme  
opérateur



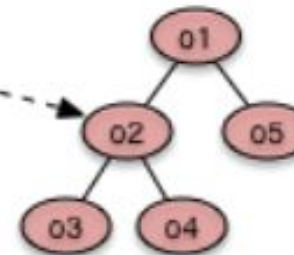
Résultat

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

```
select a1, a2, ...  
from T1, T2, ...  
where ...
```

Forme  
déclarative

?



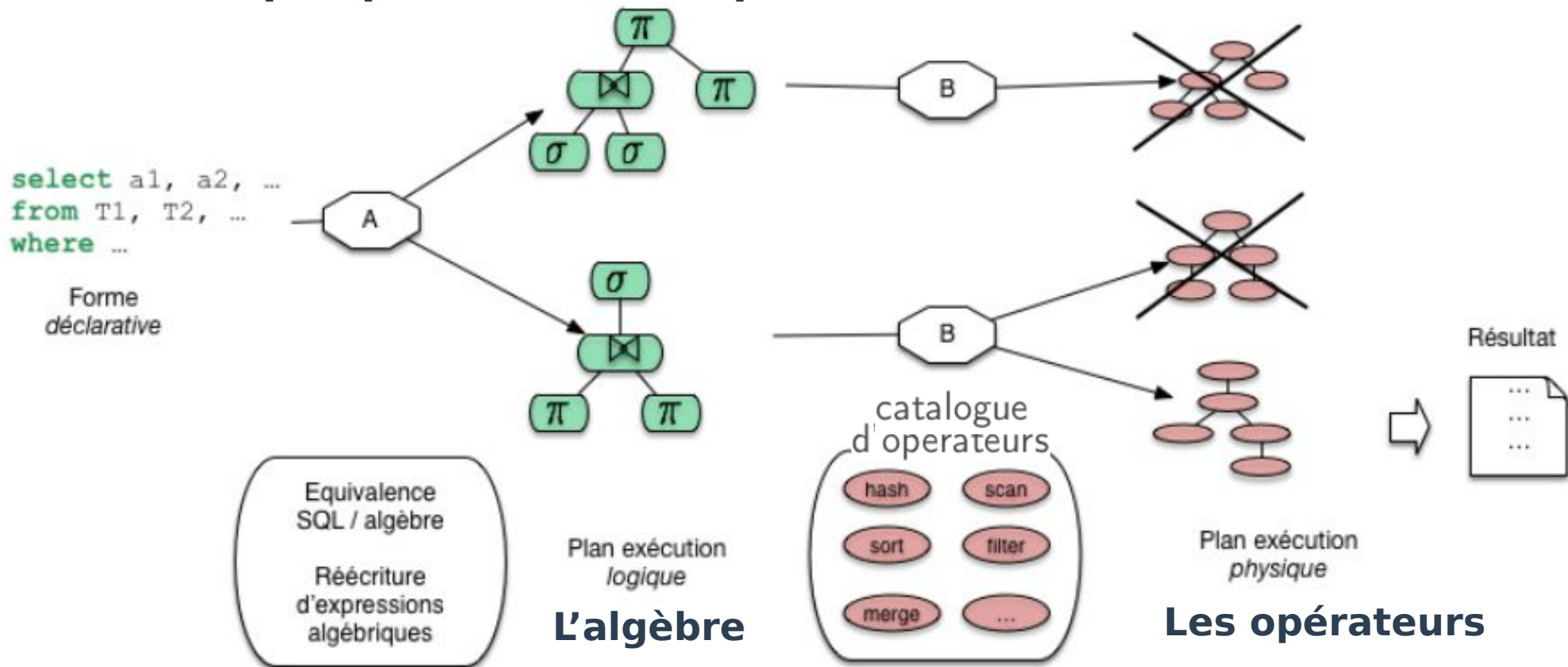
Forme  
opérateur



Résultat

# 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

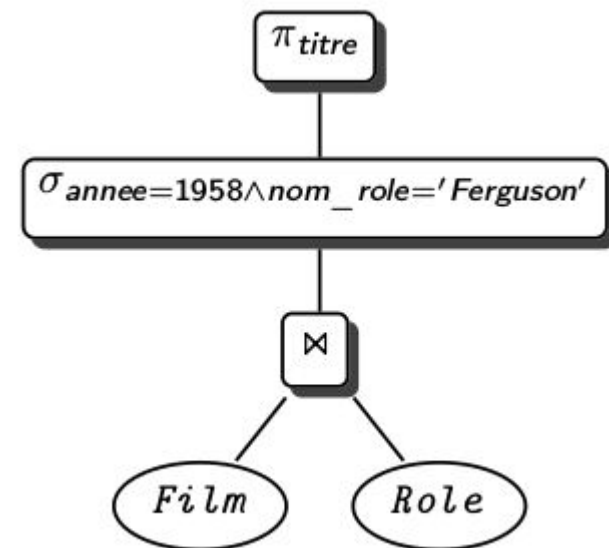


## Plan d'exécution logique (l'algèbre)

```
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

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





# Un exemple

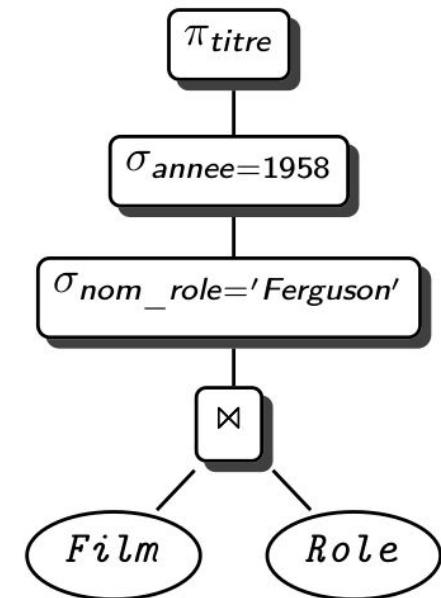
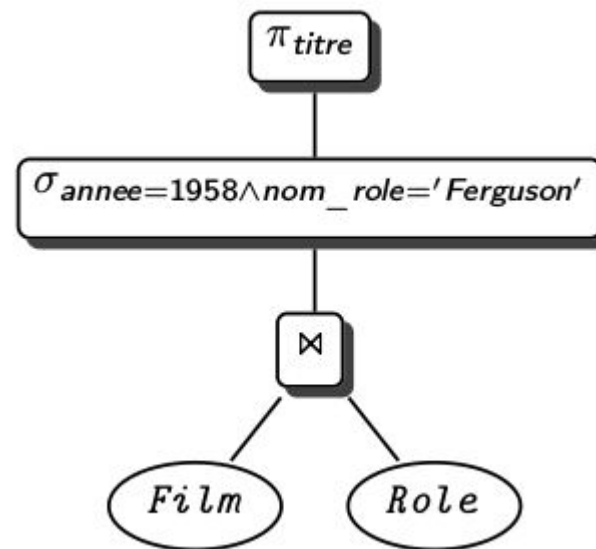
- Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.

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

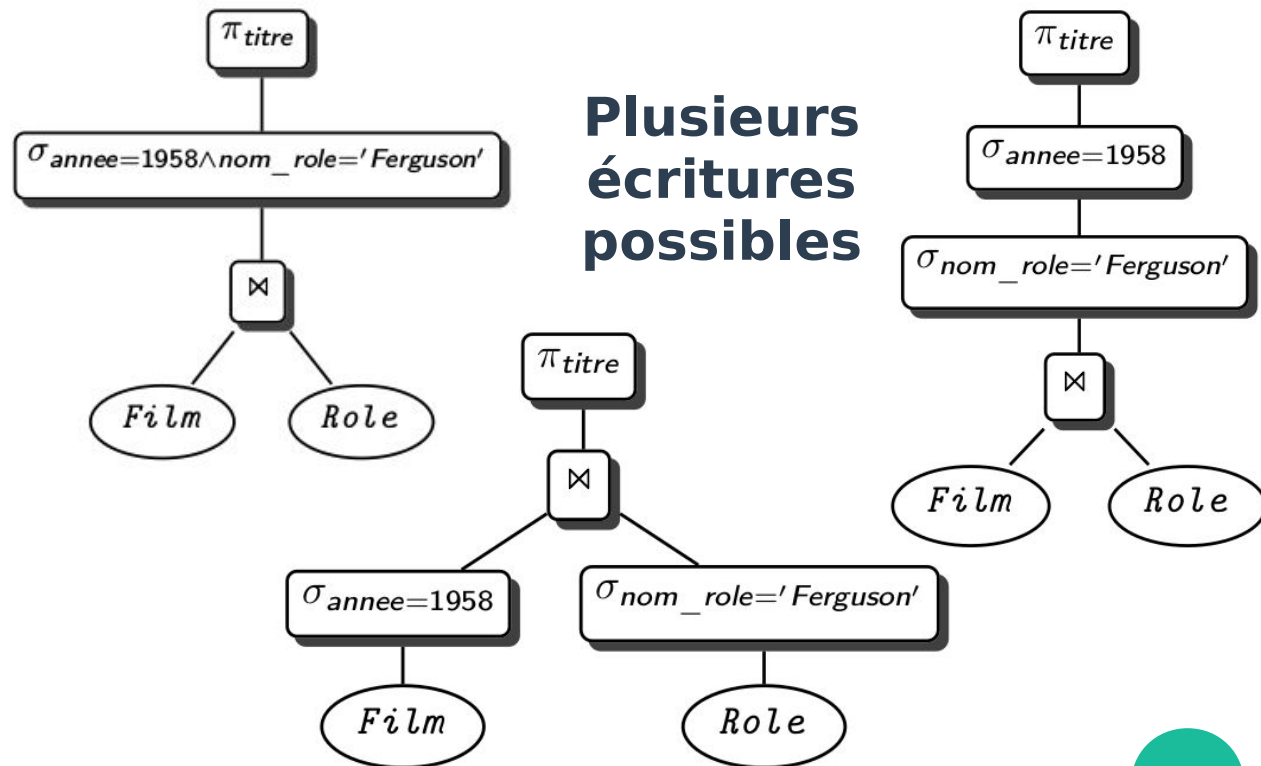
- Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.

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
```



# Un exemple

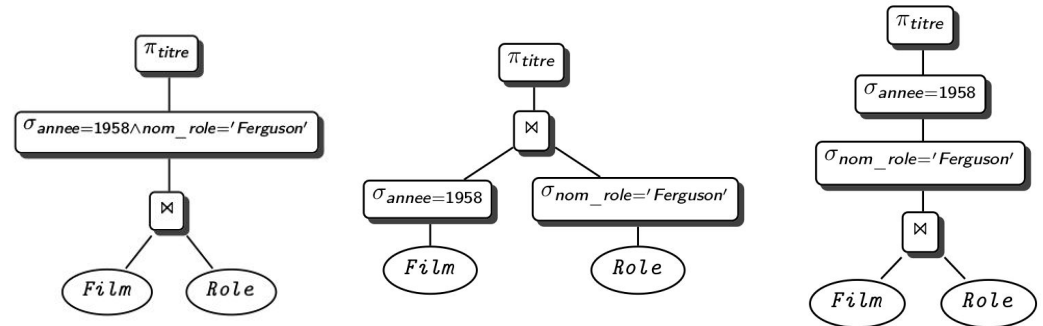
- Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.

## Requête SQL



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

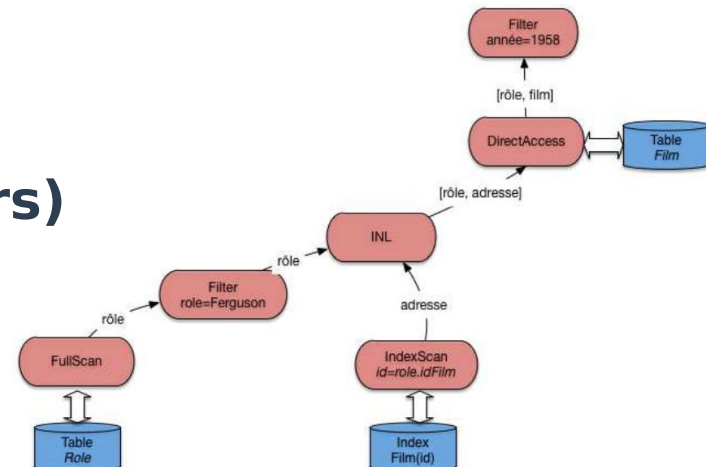
## Plan d'exécution logique (l'algèbre)



## Plan d'exécution physique (opérateurs)

Un opérateur = une opération

Plusieurs algorithmes par opération



# Un exemple

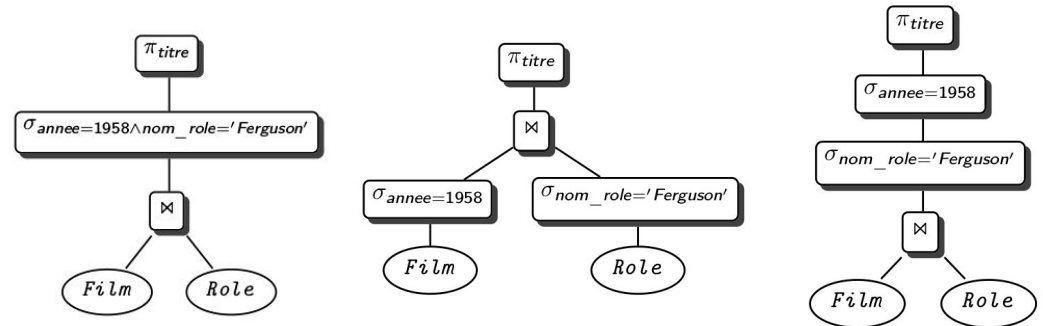
- Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.

## Requête SQL



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

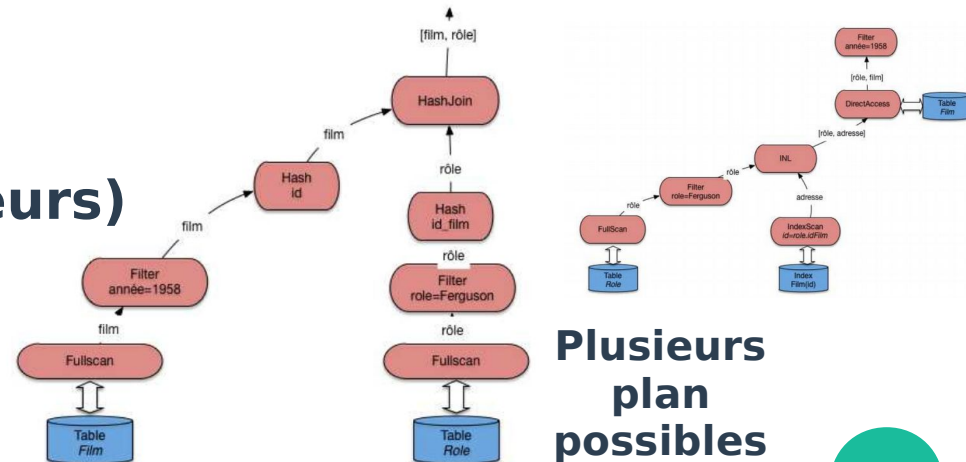
## Plan d'exécution logique (l'algèbre)



## Plan d'exécution physique (opérateurs)

Un opérateur = une opération

Plusieurs algorithmes par opération



Plusieurs  
plan  
possibles

# Un exemple

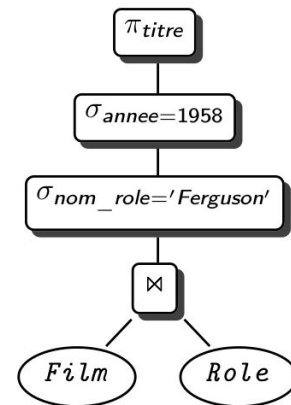
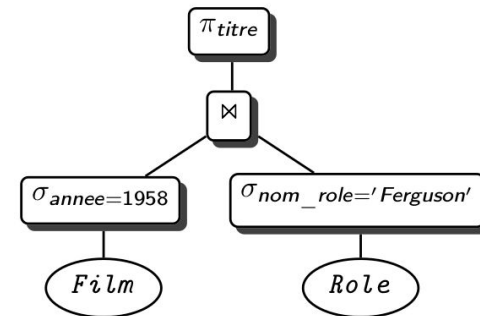
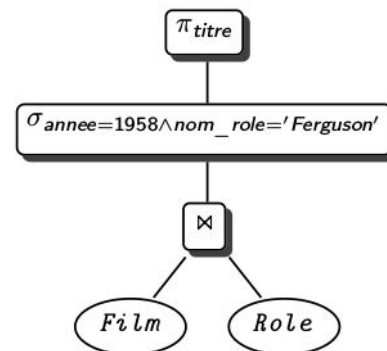
- Titre des films parus en 1958, où l'un des acteurs joue le rôle de John Ferguson.

Requête SQL



Plan d'exécution logique (l'algèbre)

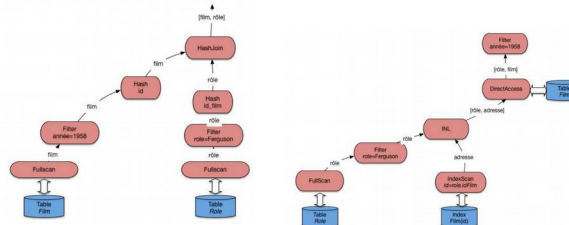
```
select titre
from Film f, Role r
where nom_role = 'Ferguson',
and f.id = r.id_ilm
and f.annee = 1958
```



Plan d'exécution physique (opérateurs)

Un opérateur = une opération

Plusieurs algorithmes par opération



Évaluation des coûts → La meilleure solution

# Le rôle de l'optimiseur

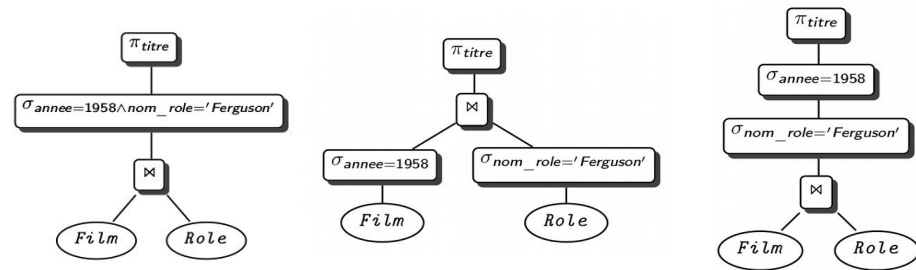
Trouver les expressions  
équivalentes

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
```

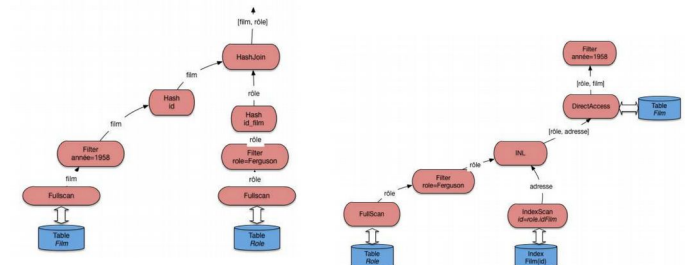


Choisir le bon algorithme  
pour chaque opération

Plan d'exécution physique (opérateurs)

Un opérateur = une opération

Plusieurs algorithmes par opération



Évaluer le coût et  
choisir la meilleure solution

# Le rôle de l'optimiseur : les expressions équivalentes

- **Problème :**

- suivant l'ordre des opérateurs algébriques dans un arbre, le coût d'exécution est différent

- **Pourquoi?**

- le coût des opérateurs varie 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 : les expressions équivalentes

- **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 de chaque plan

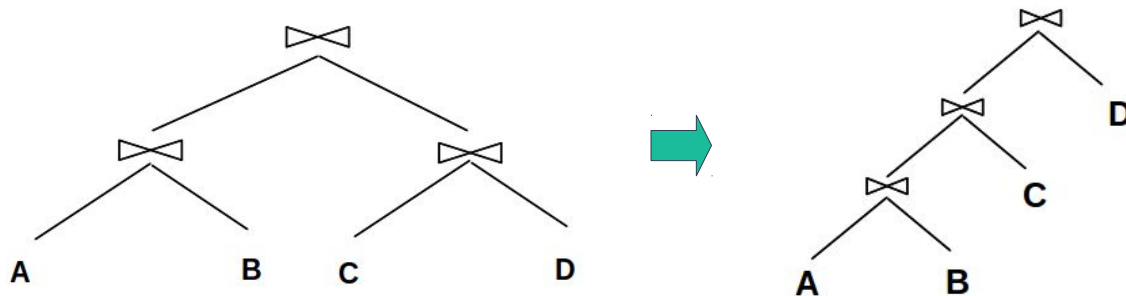
## Exemples de règles de réécriture

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  $\sigma$  et de  $\pi$**  :  $\pi_{A_1, A_2, \dots, A_p}(\sigma_{A_i='a'}(R)) \equiv \sigma_{A_i='a'}(\pi_{A_1, A_2, \dots, A_p}(R))$
4. **Commutativité de  $\sigma$  et de  $\bowtie$**  :  $\sigma_{A='a'}(R[\dots A \dots] \bowtie S) \equiv \sigma_{A='a'}(R) \bowtie S$



# Le rôle de l'optimiseur : les expressions équivalentes

- **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 côté de l'arbre (deep-left plan)

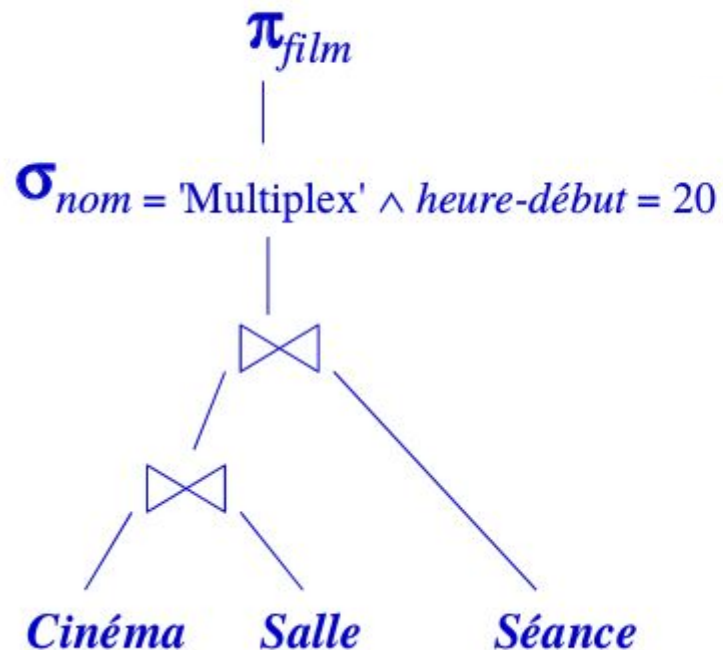


# Un exemple d'expressions équivalentes

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

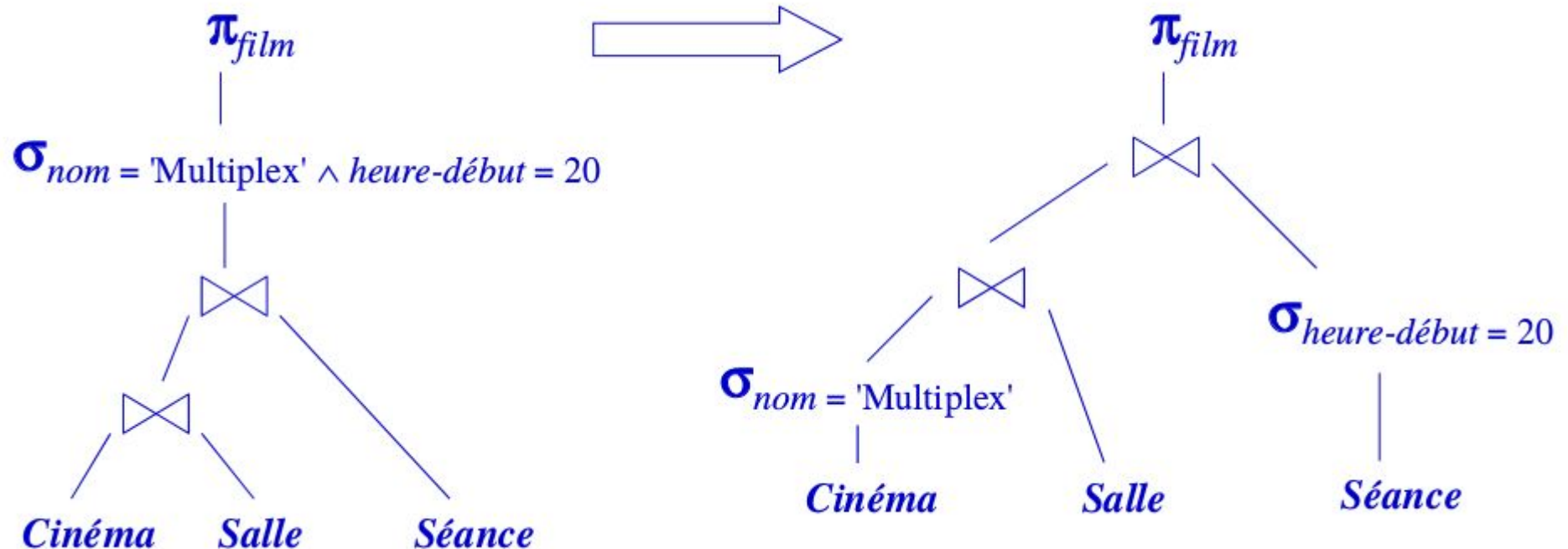
# Un exemple d'expressions équivalentes

- Arbre de requête



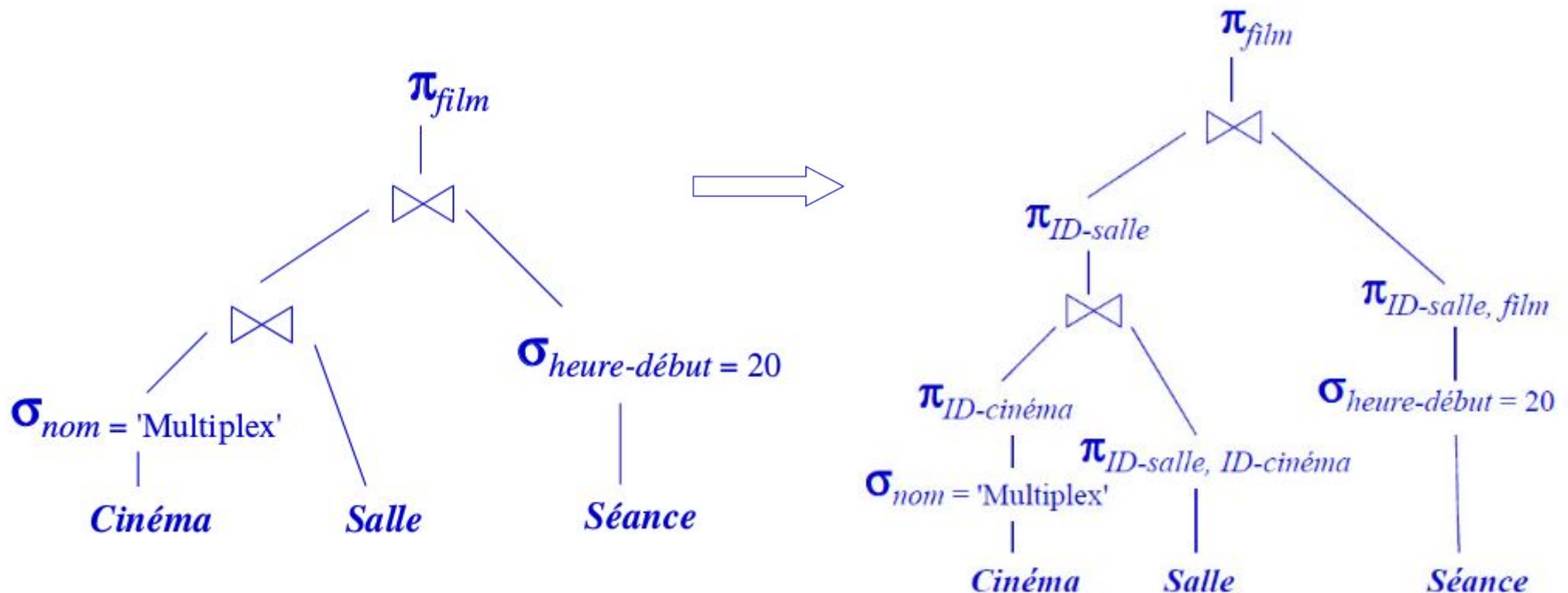
# Un exemple d'expressions équivalentes

- Arbre de requête



# Un exemple d'expressions équivalentes

- Arbre de requête



# Le rôle de l'optimiseur

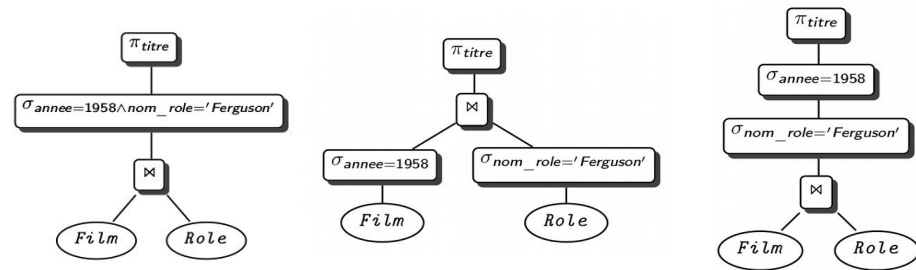
Trouver les expressions  
équivalentes

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
```

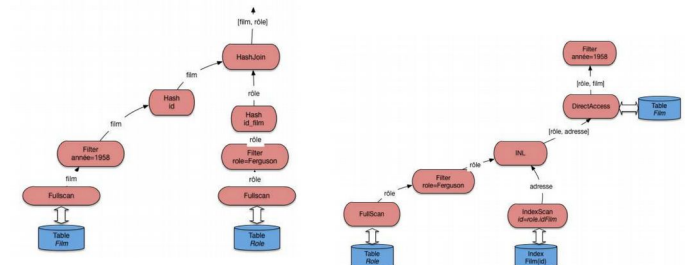


Choisir le bon algorithme  
pour chaque opération

Plan d'exécution physique (opérateurs)

Un opérateur = une opération

Plusieurs algorithmes par opération



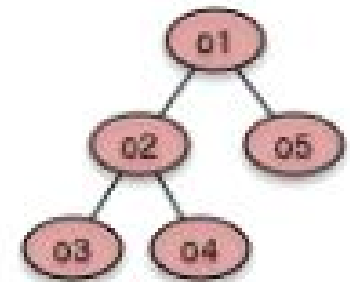
Évaluer le coût et  
choisir la meilleure solution

# Le rôle de l'optimiseur : les opérateurs

- **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

- **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

- **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

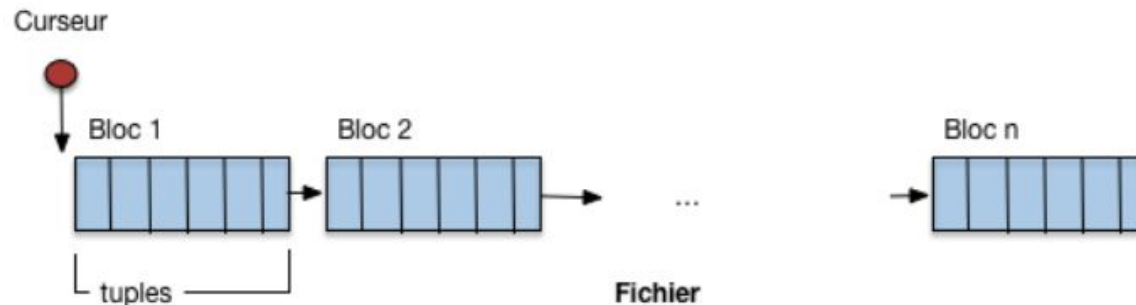
```
| select min(date) from T
```

# Le rôle de l'optimiseur : les opérateurs

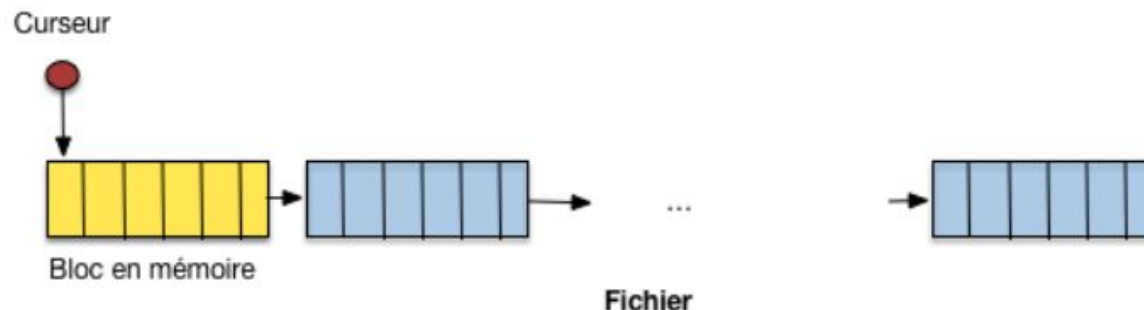
- **3 principaux types d'opérateurs et plusieurs algorithmes pour chaque opération:**
  - Accès aux données (via les tables et les index)
    - parcours séquentiel de la table (*FullScan*)
    - parcours d'index (*IndexScan*)
    - accès par adresse (*DirectAccess*)
    - test de la condition (*Filter*)
  - 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*

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

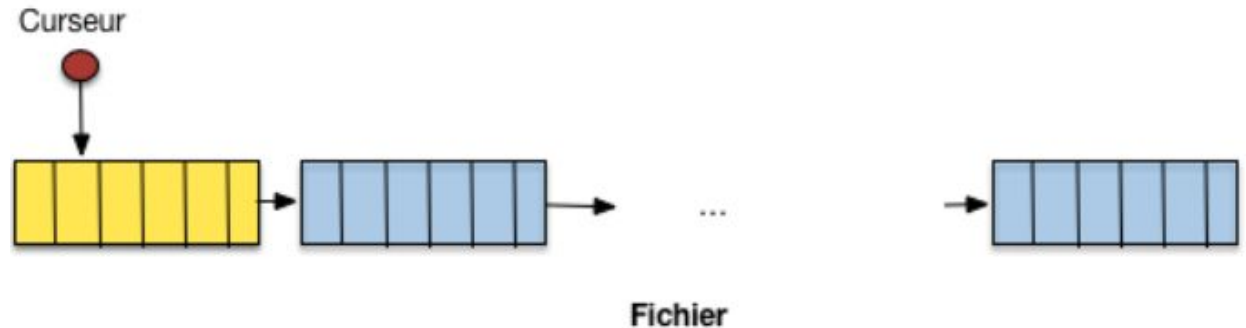


- **1<sup>er</sup> next() = accès au premier bloc, placé en mémoire**
  - Le curseur se place sur le premier nuplet, qui est retourné comme résultat. Le temps de réponse est minimal.

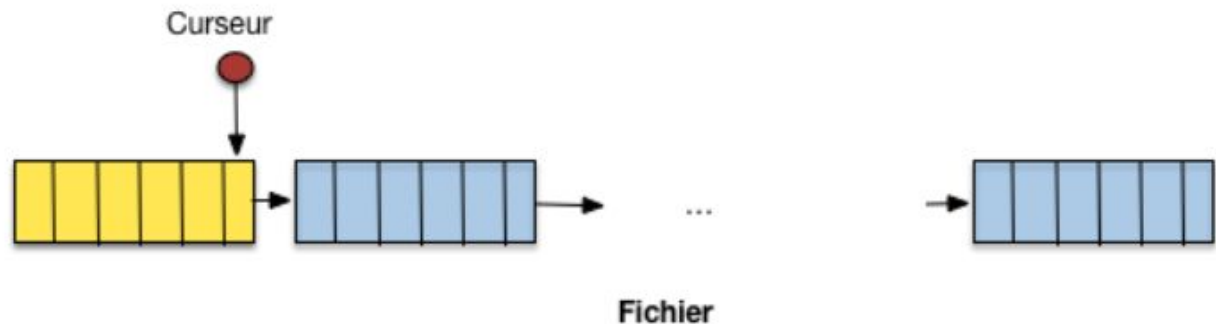


# Le rôle de l'optimiseur : l'opérateur d'accès aux données *FullScan*

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



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



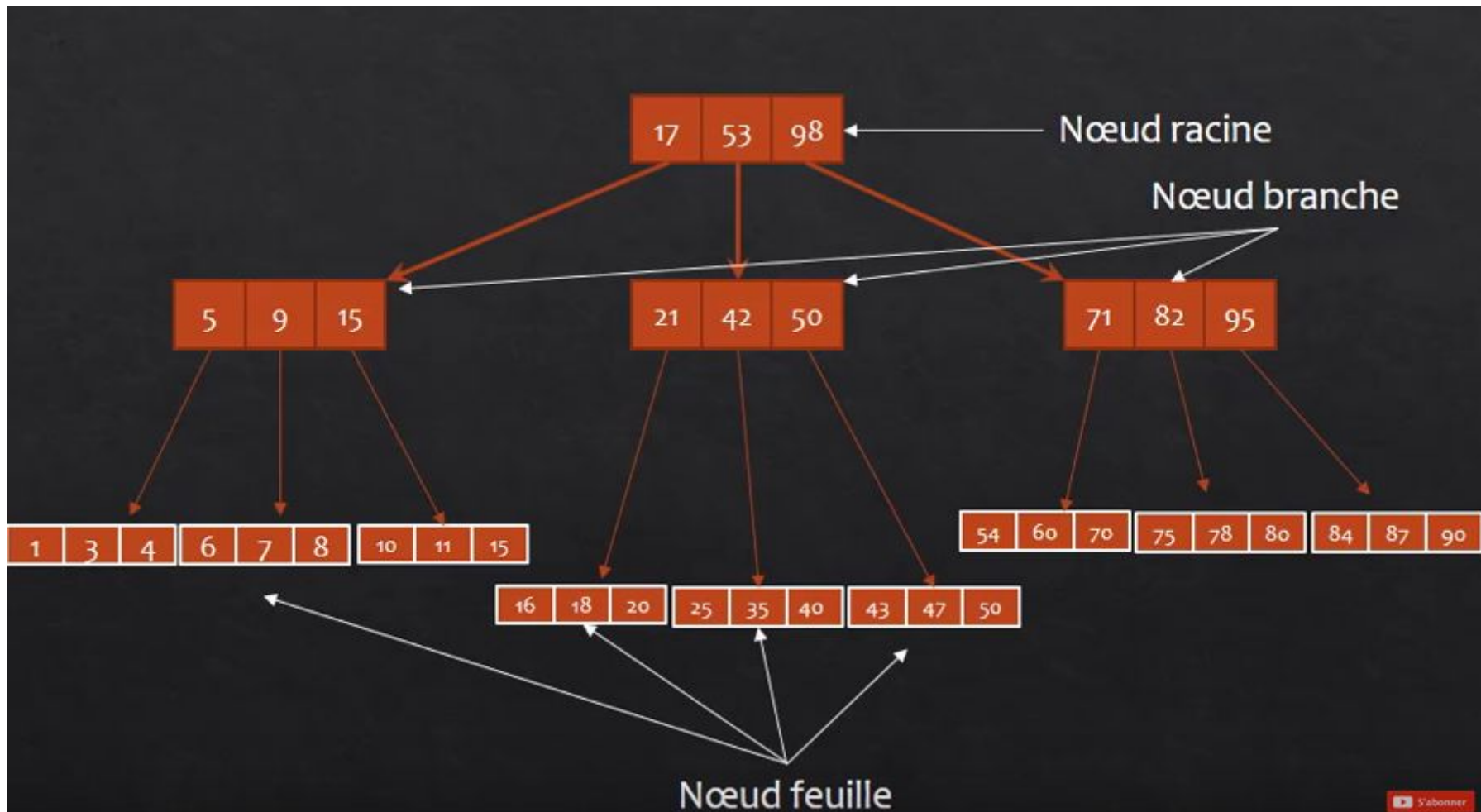
- Appel suivant à next() = charge du second bloc en mémoire



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

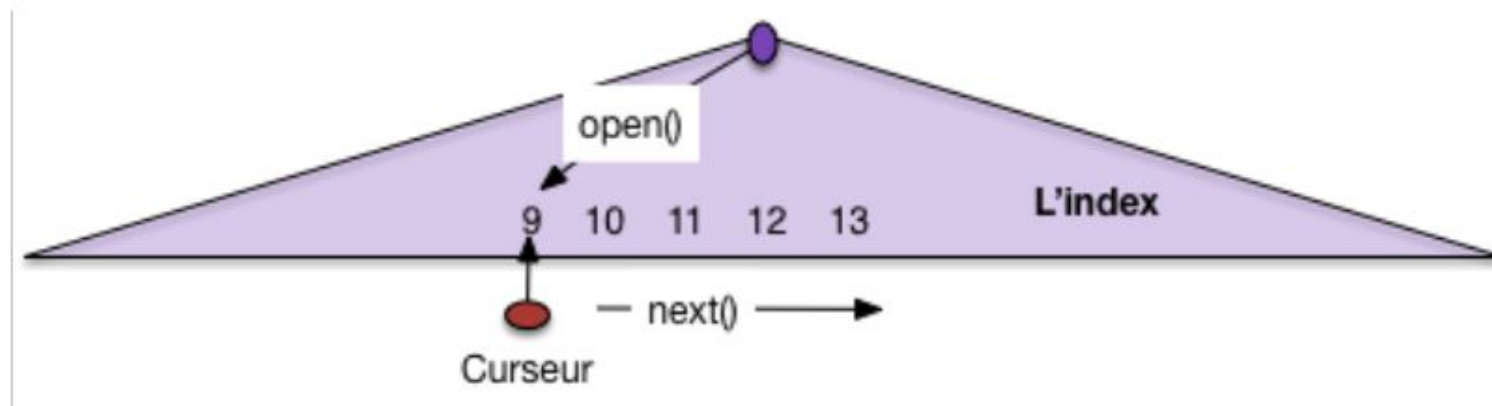
# Le rôle de l'optimiseur : l'opérateur d'accès aux données *IndexScan*

- Index = arbre équilibré (B tree)



# Le rôle de l'optimiseur : l'opérateur d'accès aux données *IndexScan*

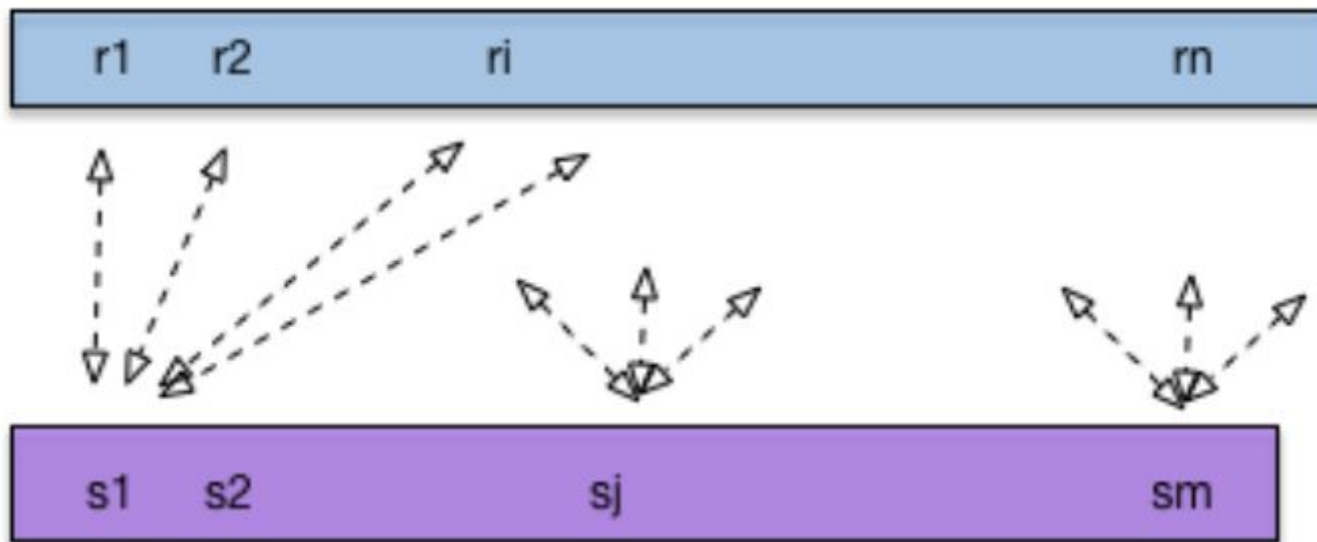
- Index = arbre équilibré (B tree)
- Pendant le `open()` : parcours de la racine vers la feuille
- À chaque appel à `next()` : parcours en séquence des feuilles



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

# Le rôle de l'optimiseur : l'opérateur de jointure *Nested loop join*

- 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)

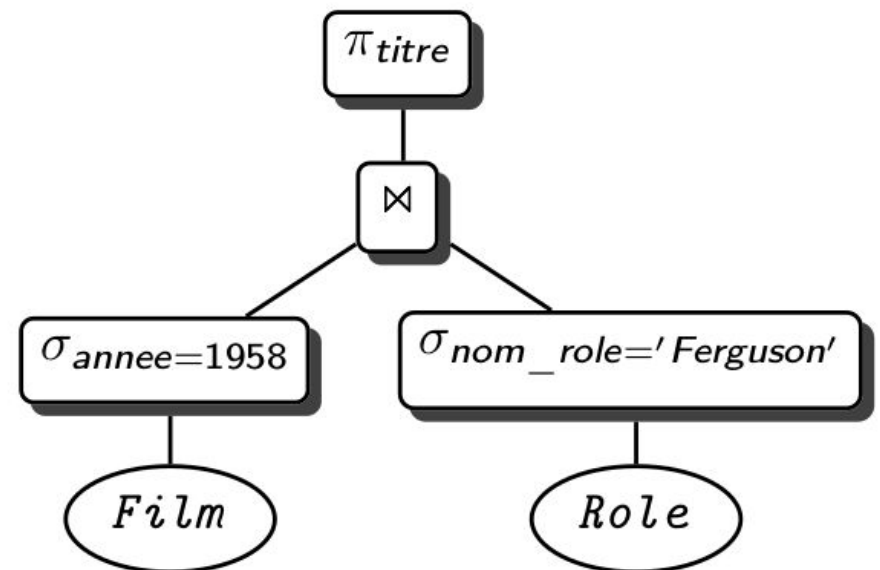
# Le rôle de l'optimiseur : l'opérateur de jointure *Hash Join*

- **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.



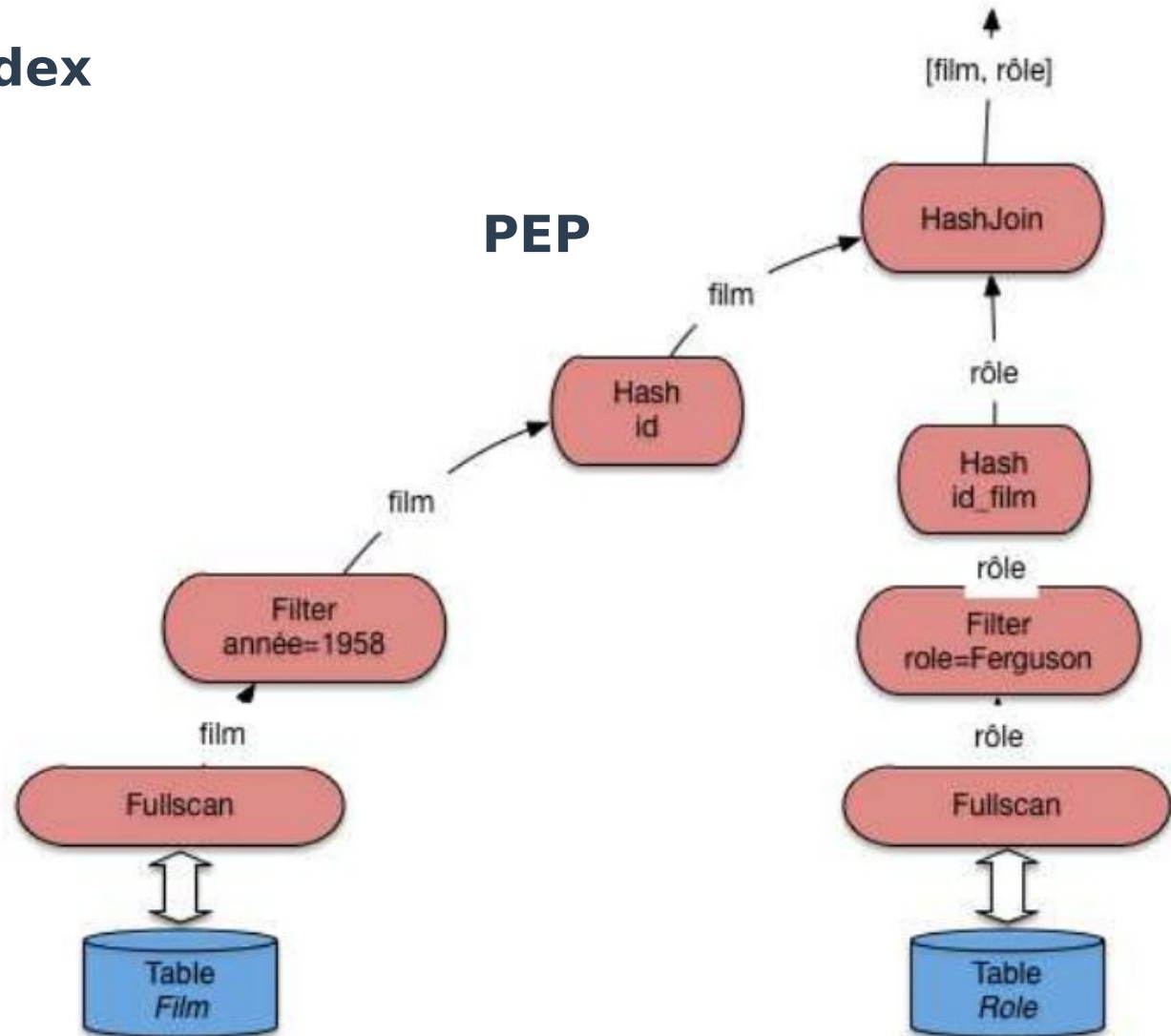
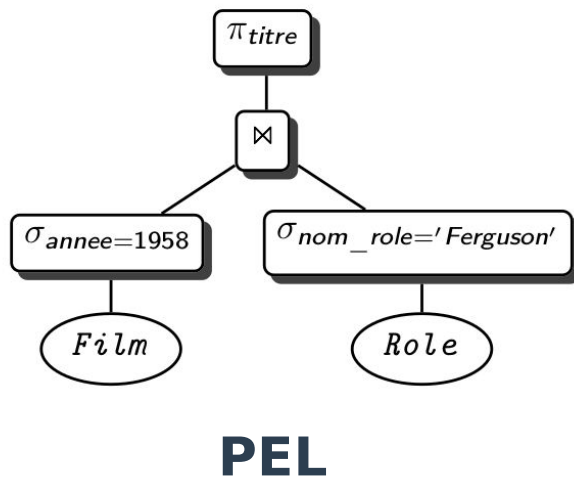
# Un exemple de plan d'exécution physique

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



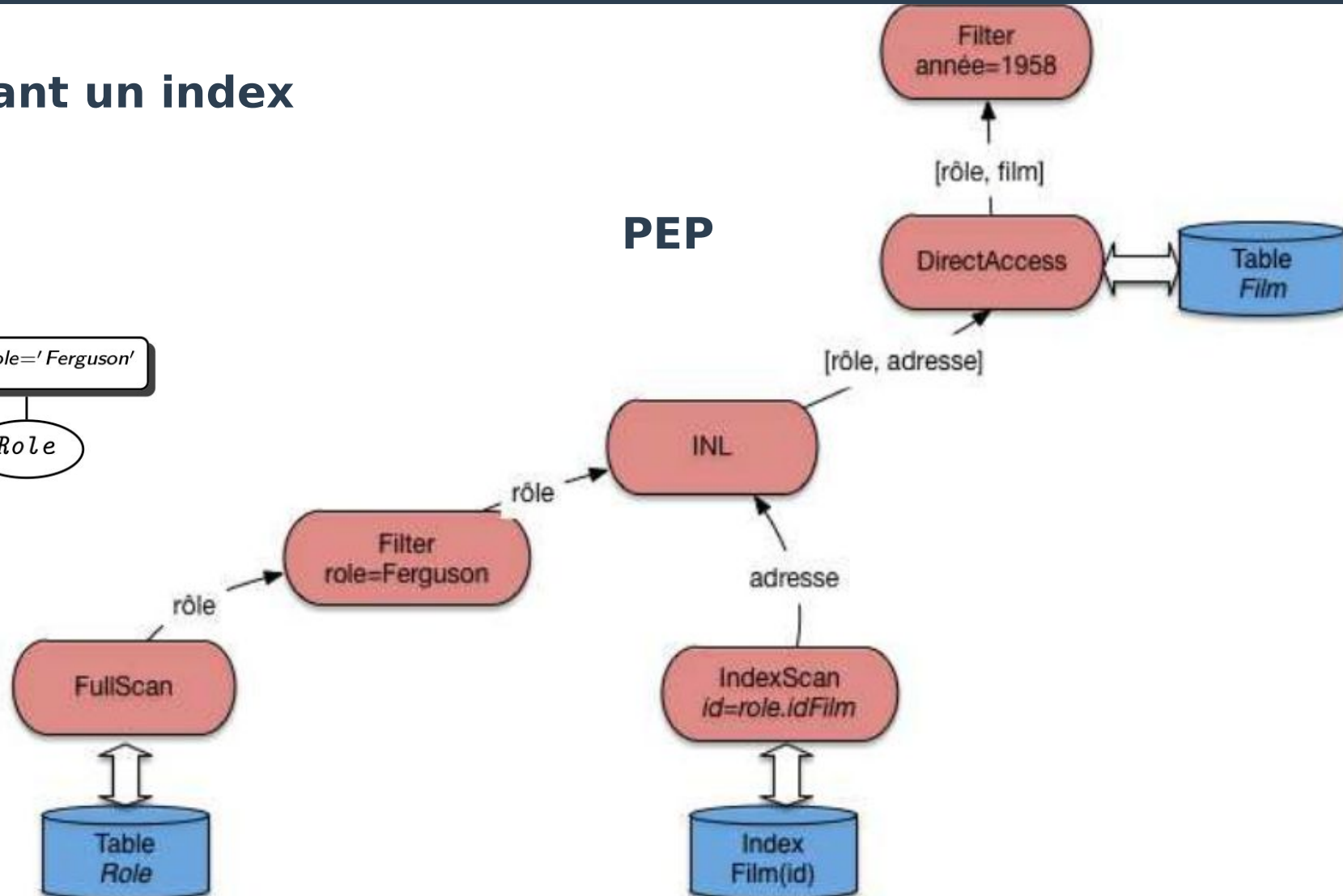
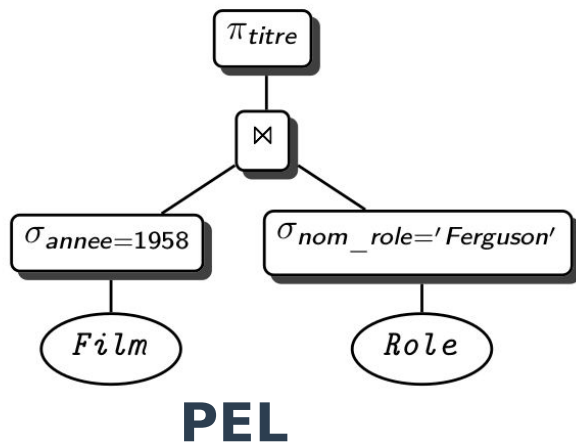
# Un exemple de plan d'exécution physique

- Sans utiliser d'index



# Un exemple de plan d'exécution physique

- En utilisant un index



# Le rôle de l'optimiseur

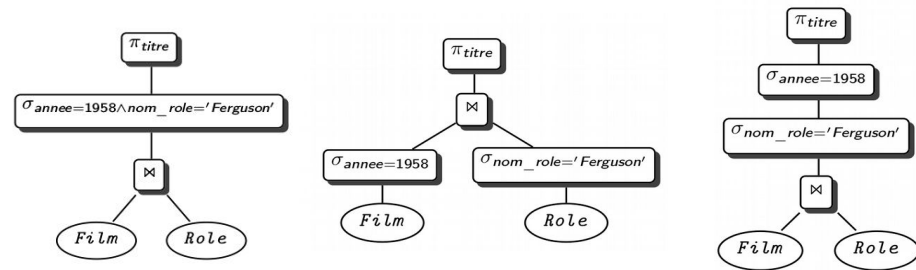
Trouver les expressions  
équivalentes

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
```

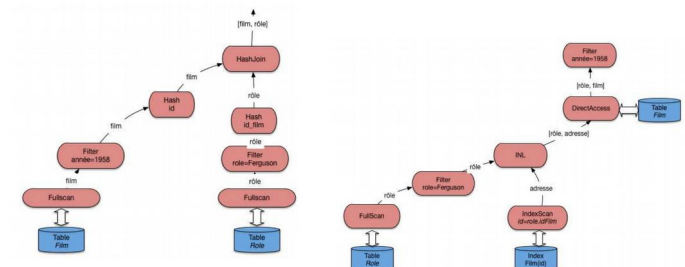


Choisir le bon algorithme  
pour chaque opération

Plan d'exécution physique (opérateurs)

Un opérateur = une opération

Plusieurs algorithmes par opération



Évaluer le coût et  
choisir la meilleure solution

# Le rôle de l'optimiseur : le coût et la meilleure solution

- **Le coût correspond au temps nécessaire pour obtenir le résultat d'une requête : plusieurs facteurs (accès disques, CPU, échange réseau)**
- **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 : le coût et la meilleure solution

- **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 : le coût et la meilleure solution

## 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`

➡  $S = 1 / \text{CARD}(A)$  avec un modèle uniforme

# Le rôle de l'optimiseur : le coût et la meilleure solution

## Sélectivité des Restrictions

$TAILLE(\sigma(R)) = s * TAILLE(R)$  avec :

- $\square s(A = \text{valeur}) = 1 / \text{CARD}(A)$
- $\square s(A > \text{valeur}) = (\max(A) - \text{valeur}) / (\max(A) - \min(A))$
- $\square s(A < \text{valeur}) = (\text{valeur} - \min(A)) / (\max(A) - \min(A))$
- $\square s(A \text{ IN liste valeurs}) =$   
 $(1/\text{CARD}(A)) * \text{CARD}(\text{liste valeurs})$
- $\square s(P \text{ et } Q) = s(P) * s(Q)$
- $\square s(P \text{ ou } Q) = s(P) + s(Q) - s(P) * s(Q)$
- $\square s(\text{not } P) = 1 - s(P)$

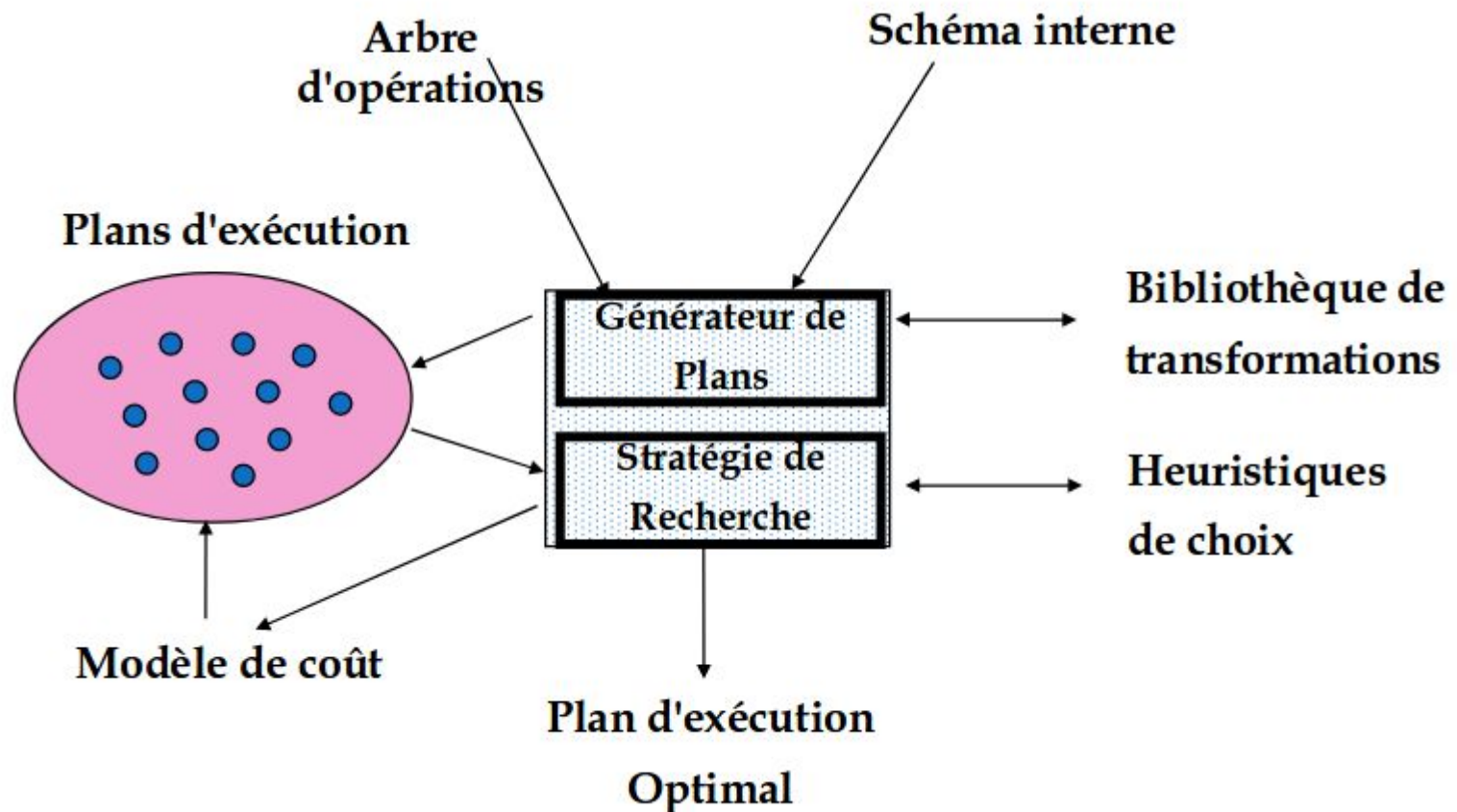


# Le rôle de l'optimiseur : le coût et la meilleure solution

- **$TAILLE(R1 \bowtie R2) = p * TAILLE(R1) * TAILLE(R2)$** 
  - $p$  dépend du type de jointure et de la corrélation des colonnes :
    - $p = 0$  si aucun  $n$ -uplet n'est joint
    - $p = 1 / \text{MAX}(\text{CARD}(A), \text{CARD}(B))$  si distribution uniforme équiprobable des attributs  $A$  et  $B$  sur un même domaine (col. Join.)
    - $p = 1$  si produit cartésien
- **L'algorithme utilisé change radicalement les coûts**
  - linéaire si index,
  - produit des tailles si boucles imbriquées

# Le rôle de l'optimiseur : le coût et la meilleure solution

- Choix du meilleur plan



# Le rôle de l'optimiseur : résumé...

- **Un plan d'exécution :**

- 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.
- 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 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 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 : 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,...)**