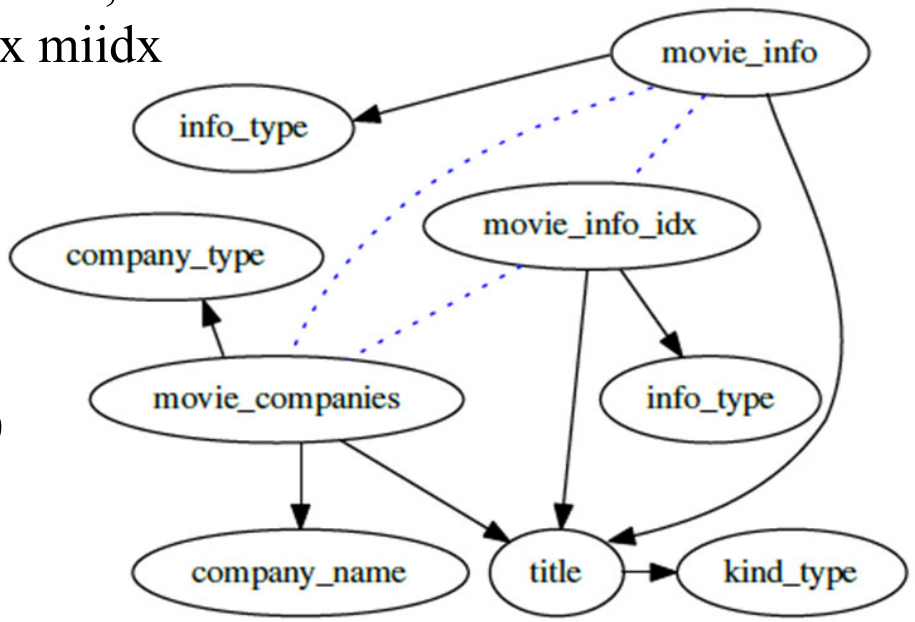
SAM 4I803
Cours 5 : Traitement et optimisation de requêtes

Intro

- La base de films IMDB
  - https://www.imdb.com/interfaces/
- Benchmark réaliste
  - Ref: How Good Are Query Optimizers, Really?
    - VLDB 2015 http://www.vldb.org/pvldb/vol9/p204-leis.pdf
  - Table *Cast_info*: 36M tuples
  - Table *Movie_info*: 15M tuples
  - Données du benchmark
    - ftp://ftp.fu-berlin.de/pub/misc/movies/database/

Requête réelle

SELECT cn.name, mi.info, miidx.info
FROM company_name cn, company_type ct,
info_type it, info_type it2, title t,
kind_type kt, movie_companies mc,
movie_info mi, movie_info_idx miidx
WHERE cn.country_code =’[us]’
AND ct.kind = ’production companies’
AND it.info = ’rating’
AND it2.info = ’release dates’
AND kt.kind = ’movie’
AND *prédicats de jointure (11 égalités)*



Toutes les requêtes: http://db.in.tum.de/people/sites/leis/qo/job.tgz

Questions

- Cardinalité
  - Estimations précises → requête rapide ?
  - Estimations très approximatives → requêtes lentes ?
- Ordre des jointures important ?
- Important d'explorer beaucoup de plans ?
  - Forme des plans
- Mesurer la performance des plans
  - Durée moyenne, max, quantile (95%)

Problème

EMP(ENO, ENAME, TITLE)
PROJECT(PNO, PNAME, BUDGET)
WORKS(ENO, PNO, RESP, DUR)

Soit la requête

pour chaque projet de budget > 250 qui emploie plus de 2 employés, donner le nom et le titre des employés

`Comment l'exprimer en SQL ?`

Un plan d'exécution possible

```
SELECT  DISTINCT Ename, Title
        FROM    Emp, Project, Works
        WHERE   Budget > 250
        AND     Emp.Eno=Works.Eno
        AND     Project.Pno=Works.Pno
        AND     Project.Pno IN
                (SELECT Pno
                 FROM   Works
                 GROUP BY Pno
                 HAVING COUNT(*) > 2)
```

$T_1 \leftarrow$ *Lire* la table `Project` et *sélectionner* les tuples de `Budget` > 250

$T_2 \leftarrow$ *Joindre* T_1 avec la relation `Works`

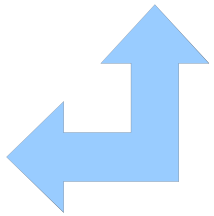
$T_3 \leftarrow$ *Joindre* T_2 avec la relation `Emp`

$T_4 \leftarrow$ *Grouper* les tuples de `Works` sur `Pno` et pour les groupes qui ont plus de 2 tuples, *projeter* sur `Pno`

$T_5 \leftarrow$ *Joindre* T_3 avec T_4

$T_6 \leftarrow$ *Projeter T5* sur `Ename`, `Title`

Résultat $\leftarrow$ *Éliminer doublons* dans T_6



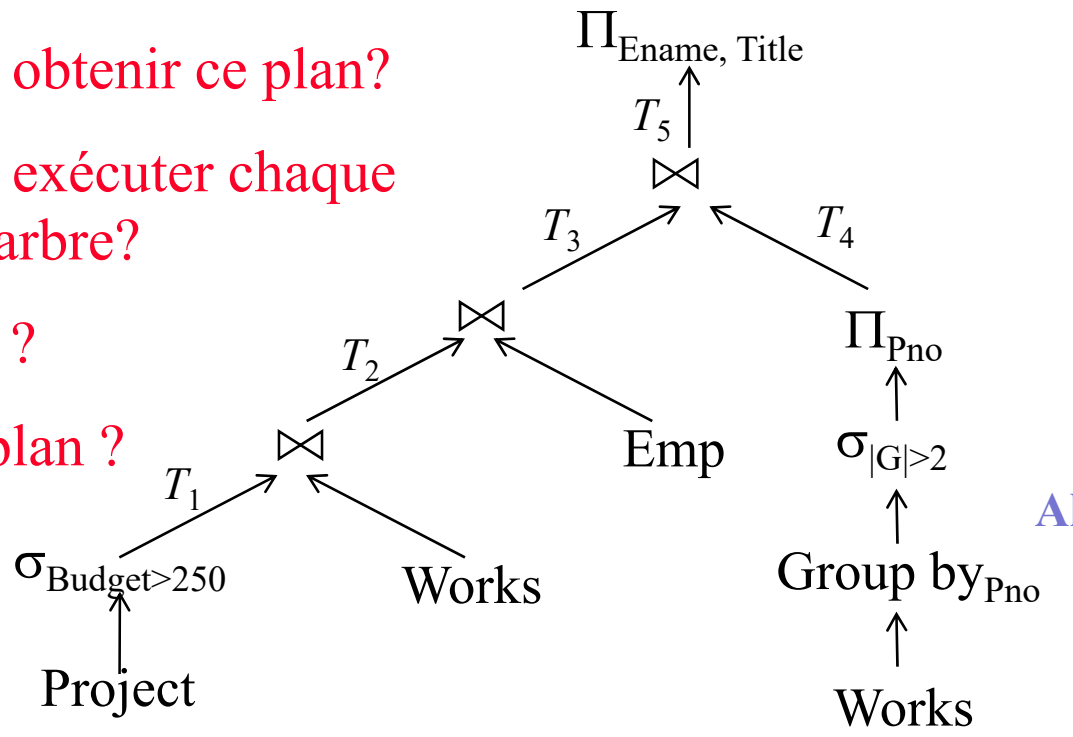
Représentation algébrique

1.Comment obtenir ce plan?

2.Comment exécuter chaque nœud/sous-arbre?

3.Quel coût ?

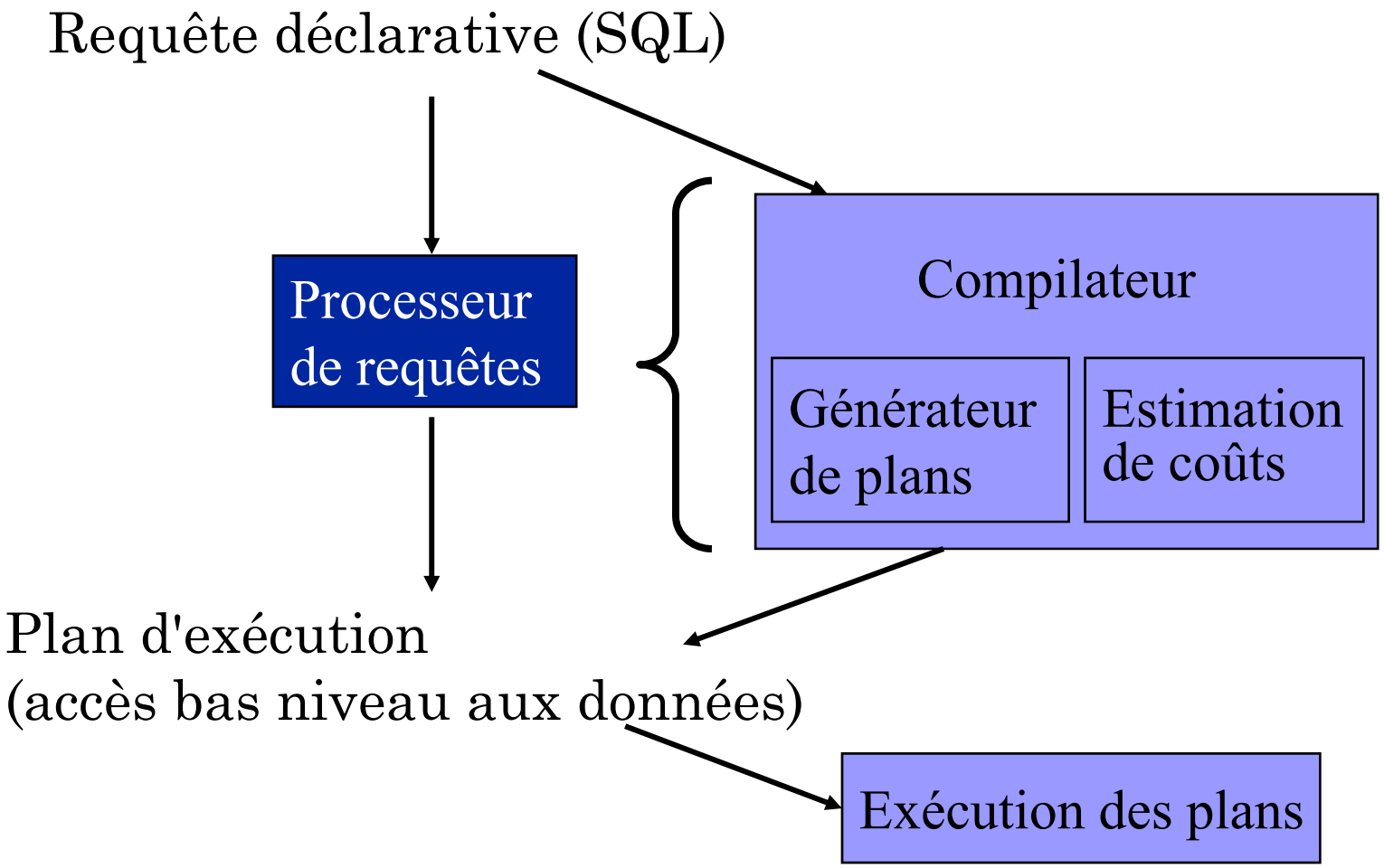
4.Meilleur plan ?



Algèbre étendue

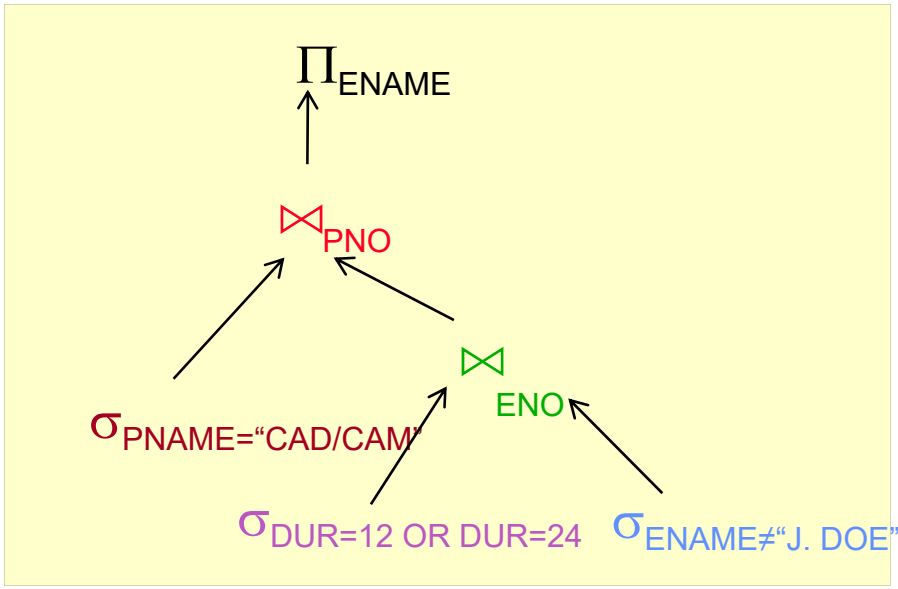
$\Pi_{Ename,\ Title}(\Pi_{Pno}(\sigma_{|G|>2}\ Group_{Pno}\ (Works)) \bowtie (Emp \bowtie ((\sigma_{Budget>250000}\ Project) \bowtie Works)))$

Rappel sur le traitement des requêtes



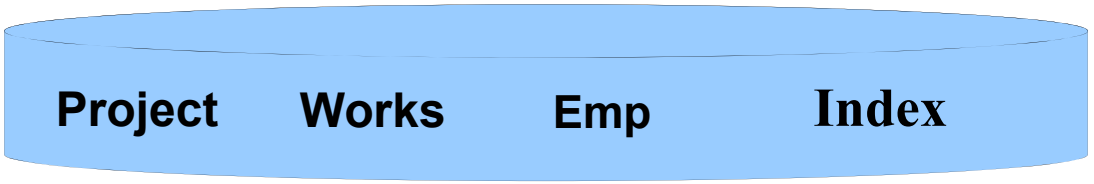
Exécution d'un plan

MÉMOIRE CENTRALE

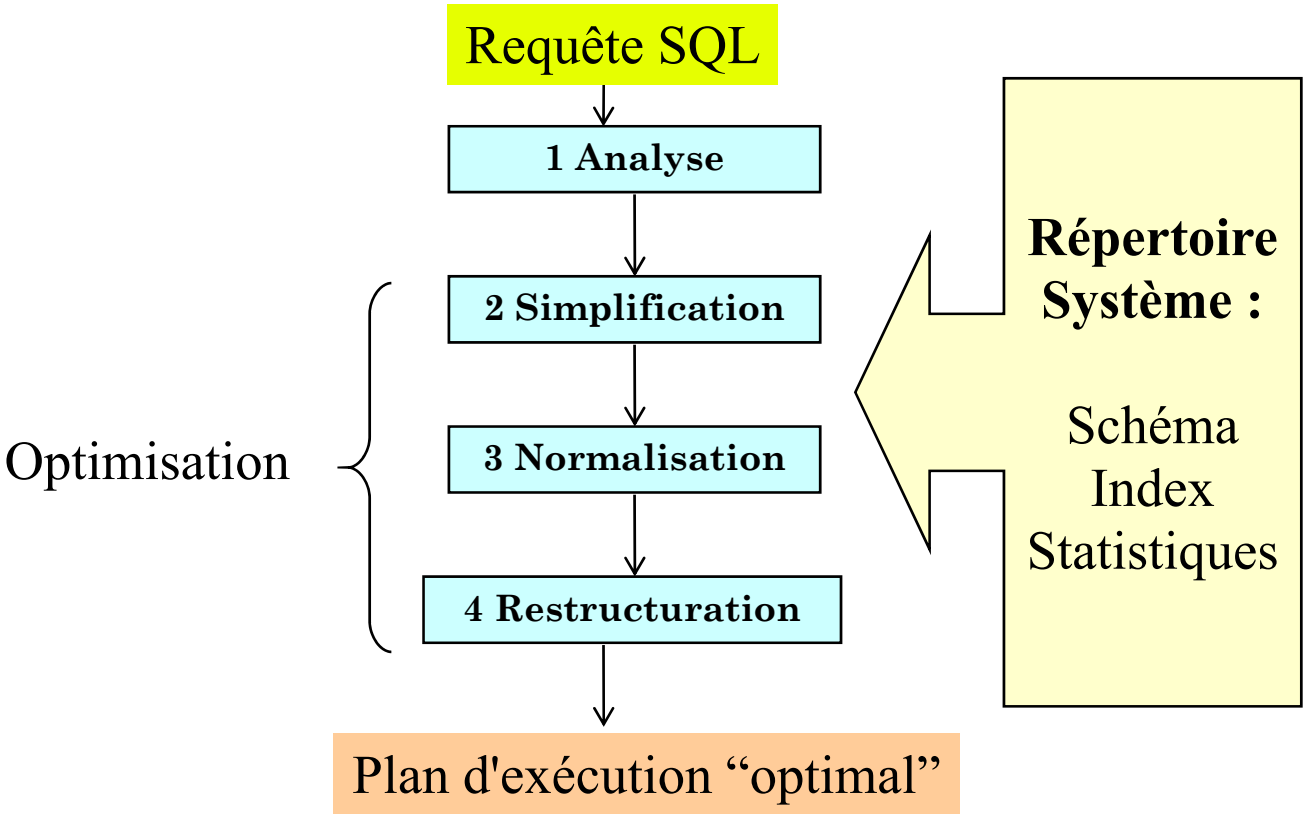


Écriture données temp. ***pages*** Lecture données / index

DISQUE



Étapes du traitement d'une requête



Normalisation de requête

- Analyse lexicale et syntaxique
  - vérification de la validité de la requête
  - vérification des attributs et relations
  - vérification du typage de la qualification
- Mise de la requête en forme normale
  - forme normale conjonctive

    $(p_{11} \vee p_{12} \vee \ldots \vee p_{1n}) \wedge \ldots \wedge (p_{m1} \vee p_{m2} \vee \ldots \vee p_{mn})$
  - forme normale disjonctive

    $(p_{11} \wedge p_{12} \wedge \ldots \wedge p_{1n}) \vee \ldots \vee (p_{m1} \wedge p_{m2} \wedge \ldots \wedge p_{mn})$
  - OR devient union
  - AND devient jointure ou sélection

Simplification

- Pourquoi simplifier?
  - plus une requête est simple, plus son exécution peut être efficace
- Comment? en appliquant des transformations
  - élimination de la redondance
    - règles d'idempotence

      $p_1 \wedge \neg(p_1) \equiv$ faux

      $p_1 \wedge (p_1 \vee p_2) \equiv p_1$

      $p_1 \vee$ faux $\equiv p_1$

      …
  - application de la transitivité (att1=att2 ,att2=att3)
- Éliminer des opérations redondantes : **élagage**
  - ex. : pas besoin de distinct après une projection sur une clé
- utilisation des règles d'intégrité
  - CI : att1 <100 Q: … where att1 > 1000…

Exemple de simplification

```
SELECT      Title
FROM        Emp
WHERE       Ename = 'J. Doe'                    P1
OR             (NOT(Title = 'Programmer')       ¬P2
AND         (Title = 'Programmer'               P2
OR             Title = 'Elect. Eng.')           P3
AND         NOT(Title = 'Elect. Eng.'))         ¬P3
```

$P1 \vee (\neg P2 \wedge (P2 \vee P3) \wedge \neg P3)$

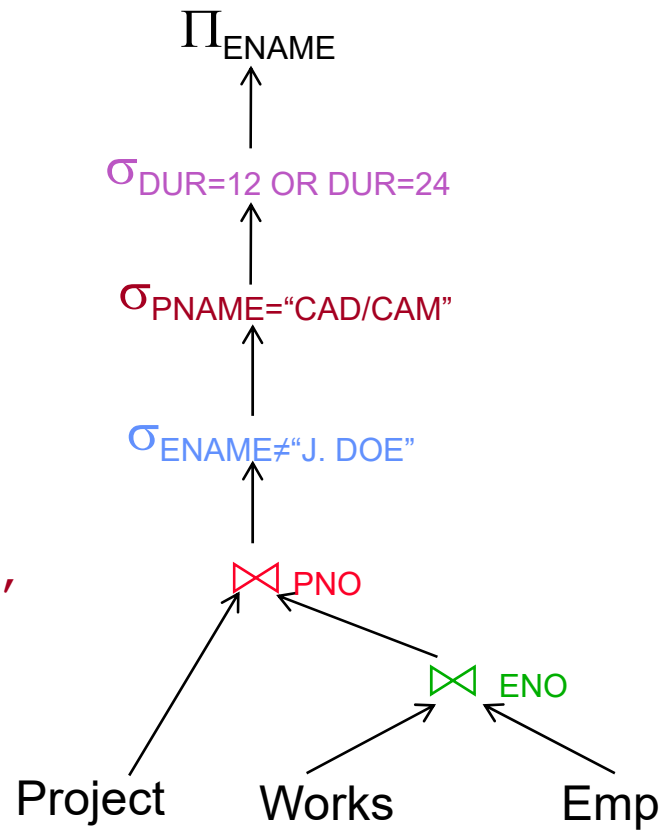
```
SELECT      Title
FROM        Emp
WHERE       Ename = 'J. Doe'
```

Traduction en algèbre

Conversion en arbre algébrique

Exemple :

```
SELECT   Ename
FROM     Emp, Works, Project
WHERE    Emp.Eno = Works.Eno
AND      Works.Pno = Project.Pno
AND      Emp.Ename <>'J.Doe'
AND      Project.name = 'CAD/CAM'
AND      (Works.Dur=12 OR
           Works.Dur=24)
```



Heuristiques

Observation : les opérations manipulant moins de données sont plus rapides (sélectivité corrélée à la performance)

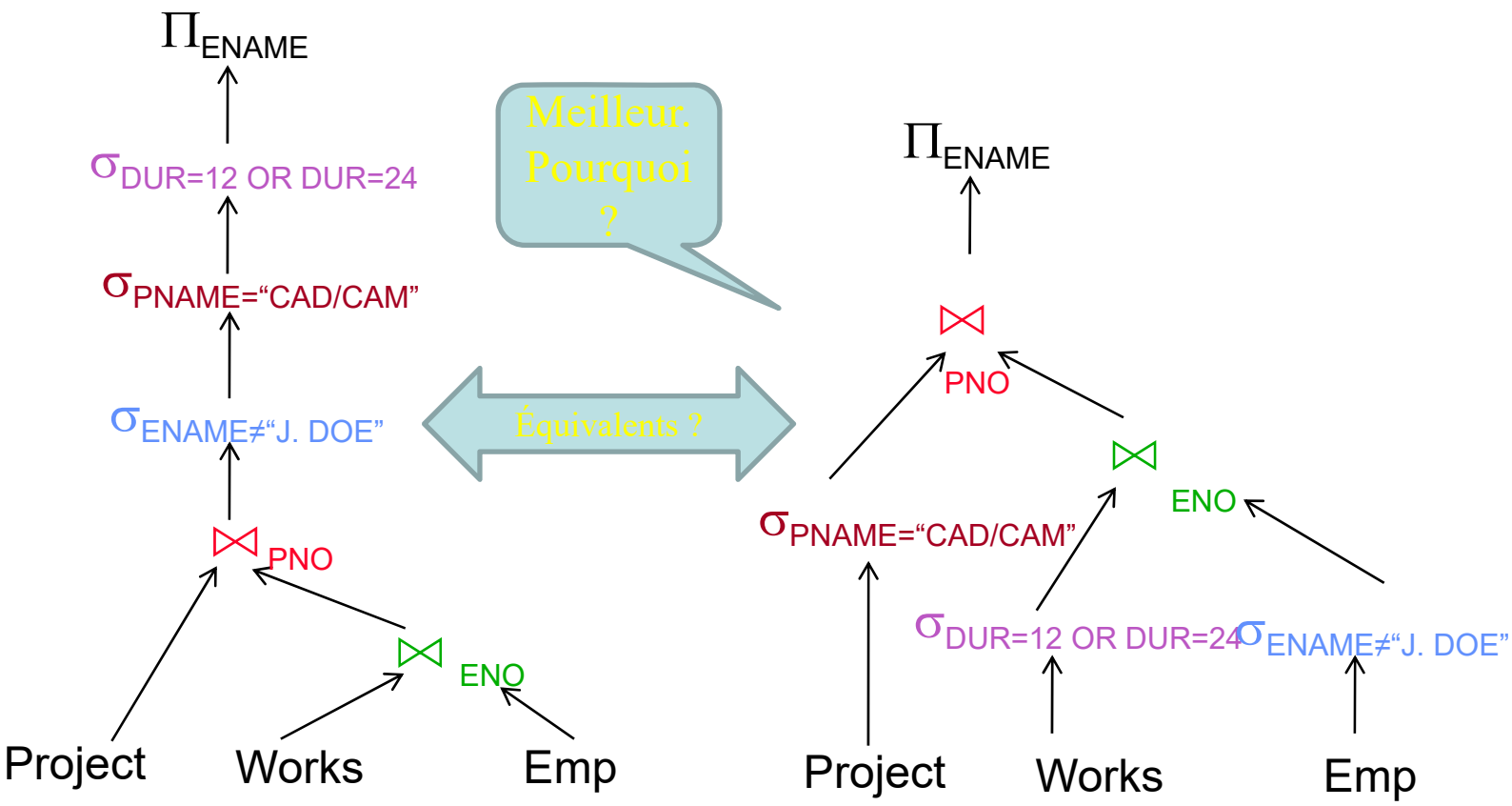
Objectif : déterminer un ordre pour les opérations, censé être efficace.

Méthode:

Commencer par traiter les opérations les **plus sélectives** (projection, sélection) et de manière à réduire la taille des données d'entrée pour les opérateurs suivants (jointures).

La place en mémoire est un facteur primordial pour l'efficacité d'une jointure (cf. algo de jointure, cours suivant)

Exemple


Π ENAME
σ DUR=12 OR DUR=24
σ PNAME="CAD/CAM"
σ ENAME≠"J. DOE"
⋈ PNO
⋈ ENO
Project
Works
Emp
Meilleur. Pourquoi ?
Équivalents ?
Π ENAME
⋈ PNO
⋈ ENO
σ PNAME="CAD/CAM"
σ DUR=12 OR DUR=24
σ ENAME≠"J. DOE"
Project
Works
Emp

Optimisation basée sur le coût

- Elaborer des plans
  - arbre algébrique, restructuration, ordre d'évaluation
- Estimer leurs coûts
  - fonctions de coût
    - en terme de temps d'exécution
    - coût I/O + coût CPU
    - poids très différents
      - par ex. coût I/O = 1000 * coût CPU
- Choisir le meilleur plan
  - Espace de recherche : ensemble des expressions algébriques équivalentes pour une même requête
  - algorithmes de recherche:
    - parcourir l'espace de recherche
    - algorithmes d'optimisation combinatoire

Restructuration

- Objectif : choisir l'ordre d'exécution des opérations algébriques (élaboration du plan logique).

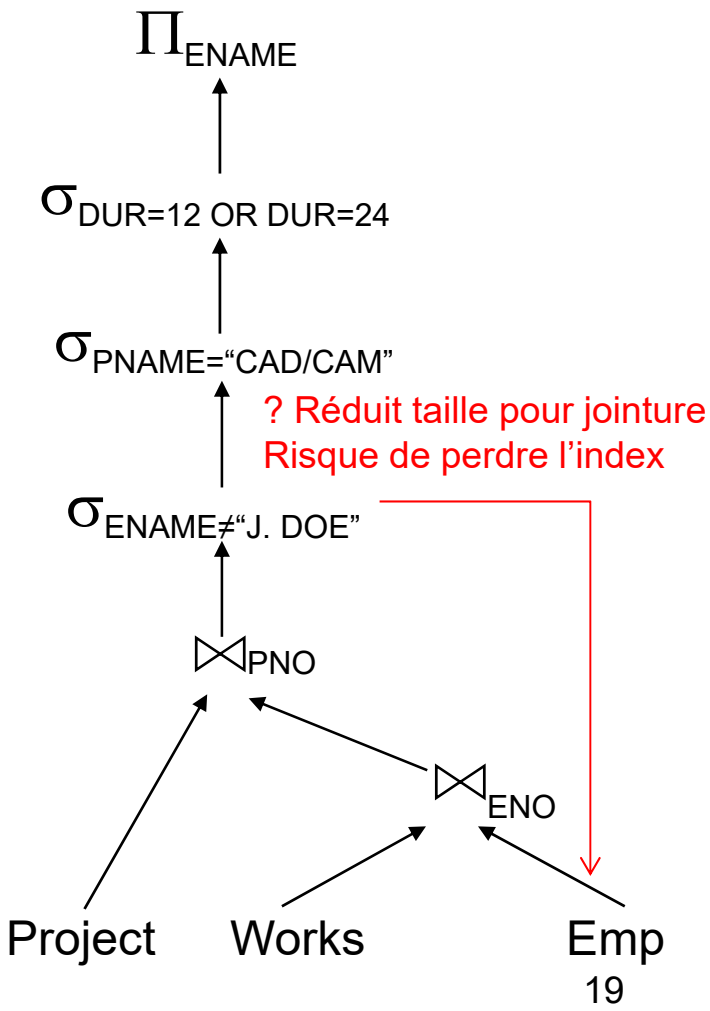
- Conversion en arbre algébrique

- Transformation de l'arbre (optimisation)
  - règles de transformation (équivalence algébriques),
  - estimation du coût des opérations en fonction de la taille
  - Estimation du résultat intermédiaire (taille et ordre?)
  - En déduire l'ordre des jointures

Restructuration

- Conversion en arbre algébrique
- Exemple

```
SELECT  Ename
FROM    Emp, Works, Project
WHERE   Emp.Eno=Works.Eno
AND     Works.Pno=Project.Pno
AND     Ename NOT='J.Doe'
AND     Pname = 'CAD/CAM'
AND     (Dur=12 OR Dur=24)
```



Coût d'un plan

- Fonction de coût = estimation du temps écoulé pour
  - Accéder aux données :
    - temps I/O exprimé en nombre de pages lues ou écrites
  - Calculer le résultat d'une opération à partir des données lues
    - Temps CPU dépend du nombre d'instructions
- Estimation du coût d'exécution de chaque noeud de l'arbre algébrique
  - utilisation de pipelines ou de relations temporaires importante
    - Pipeline : les tuples sont passés directement à l'opérateur suivant.
      - Pas de relations intermédiaires (petites mémoires, ex. carte à puce).
      - Permet de paralléliser (BD réparties, parallèle)
      - Intéressant même pour cas simples : $\sigma_{F \wedge F'}(R)$, index sur F' $\rightarrow \sigma_F(\sigma_{F'}(R))$
    - Relation temporaire : permet de trier mais coût de l'écriture

Estimer la cardinalité d'une opération

- Estimation du nombre de nuplets résultant de chaque nœud par rapport à ses entrées

  – Permet d'estimer le coût de l'opération suivante
  – sélectivité des opérations – "facteur de réduction"
  – propagation d'erreur possible
  – basé sur les statistiques maintenues par le SGBD

Statistiques

- Relation
  - cardinalité : card(R)
  - taille d'un tuple : largeur(R)
  - fraction de tuples participant une jointure / attribut
  - …
- Attribut
  - cardinalité du domaine
  - nombre de valeurs distinctes **D**(R,A) = $\Pi_A(R)$
  - Valeur max, valeur min
- Hypothèses
  - **Indépendance** entre différentes valeurs d'attributs
  - Distribution **uniforme** des valeurs d'attribut dans leur domaine
  - Sinon, il faut maintenir des histogrammes (voir diapo suivante)
- Stockage :
  - Les statistiques sont des métadonnées, stockées sous forme relationnelle (cf. TME)
  - Rafraîchies périodiquement, pas à chaque fois.

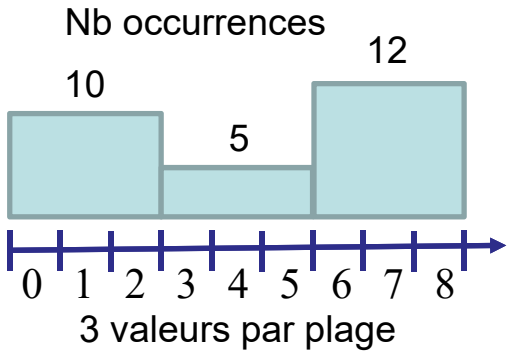
compromis L/E

Histogrammes

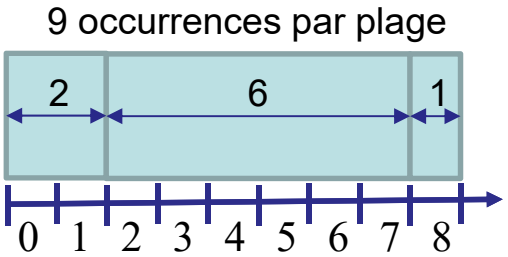
Sert à estimer la cardinalité de la requête :
Select * from R where A = 8 ?

A	card
0	7
1	2
2	1
3	5
4	0
5	0
6	2
7	1
8	9

Histogramme Equilarge
Plages de valeurs de même taille.
Hypothèse d'uniformité
dans un intervalle



Histogramme Equiprofond
Plages de valeurs contenant le même nombre d'occurrences.
Plus précis pour les valeurs fréquentes.



Card **estimée** pour A=8 ?

card = 12 / 3 = **4**

Card = **9**

Cardinalité réelle = **9**

Cardinalité d'une sélection
Facteur de sélectivité

- Soit la **sélection** notée : $\sigma_{pred}(R)$
  - *pred* est le prédicat de sélection
  - En SQL : select * from R where pred

- **Cardinalité** d'une sélection

$$card(\sigma_{pred}(R)) = SF(pred) * card(R)$$

SF est une *estimation* de la **sélectivité du prédicat** *pred*, dont la forme générale est "nombre d'éléments sélectionnés / nombre d'éléments possibles"

Facteur de sélectivité : Sélection sur un attribut

- Egalité : sélection de valeurs
  - Nbval: nombre de valeurs sélectionnées
  - **SF = Nbval / D(R,A)**

pred	Nbval
A = v	1
A in (v1, v2, ..., vk)	k
A= v1 OR A= v2 OR ... OR A= vk	k

- Inégalité : sélection d'un intervalle
  - Intervalle = segment d'un domaine (continu)
  - Bornes du domaine : A ∈[min(A), max(A)]
  - Longueur du domaine : L(A) = max(A) – min(A)
  - **SF = longueur du segment sélectionné / L(A)**

pred	Longueur
A < v	v – min(A)
A > v	max(A) – v
A ≥ v1 and A ≤ v2 s'écrit aussi: A between v1 and v2	v2 – v1
A ≤ v1 or A ≥ v2	max(A) - v2 + v1 – min(A)

Facteur de sélectivité
Sélection sur plusieurs attributs

- Le prédicat est **composé** de plusieurs prédicats:
  - $pred_A$, $pred_B$, ...
  - Exple: $pred_{age}$: age > 20 $pred_{Ville}$: ville='Paris'

- Conjonction: $pred_A$ AND $pred_B$
  - SF($pred_A$ AND $pred_B$) = $SF(pred_A) * SF(pred_B)$
  - Exple:
    - Pred : age > 20 **AND** ville = 'Paris'
    - SF(pred) = SF(age > 20) * SF(ville='Paris')

- Disjonction entre 2 termes : $pred_A$ OR $pred_B$
  - SF($pred_A$ OR $pred_B$) = $SF(pred_A) + SF(pred_B) - SF(pred_A) * SF(pred_B)$
  - Exple:
    - Pred : age >20 **OR** ville = 'Paris'
    - SF(pred) = SF(age > 20) + SF(ville = 'Paris') – SF(age > 20) * SF(ville = 'Paris')

- Rmq: peut se généraliser pour *n* prédicats
  - Disjonction n-aire: *(n-1)* disjonctions binaires

Facteur de sélectivité
Sélection avec **négation**

- Négation: complément à 1
- SF (not (pred)) = 1 – SF(pred)

- Exemple:
  - jourFermeture = 'Dimanche'
    - SF(jourFermeture = 'Dimanche') = 1/7

  - jourFermeture <> 'Dimanche'
    - SF(jourFermeture <> 'Dimanche') = 1 – 1/7 = 6/7

Cardinalité des opérations

Projection

$card(\Pi_A(R)) <= card(R)$ (égalité si A est *unique*)

Produit cartésien

$card(R \times S) = card(R) \cdot card(S)$

Union

borne sup. : $card(R \cup S) = card(R) + card(S)$

borne inf. : $card(R \cup S) = max\{card(R), card(S)\}$

Différence

borne sup. : $card(R–S) = card(R)$

borne inf. : 0

Cardinalité d'une jointure naturelle

- Jointure naturelle entre 2 tables R et S sur l'attribut A
  - A est clé "primary key" de R(A, ...)
    - donc card(R) / D(R, A) = **1**
  - A est clé étrangère "foreign key" dans S (...., A*, ...)
  - domaine(S.A) ⊆ domaine(R.A)
  - Pour chaque tuple de S, il existe **un et un seul** tuple de R

- $card(R \bowtie_A S) = card(S) . 1$

- Exple :
  - **Etu**(<u>nomE</u>, age) **Note**(nomE*, codeUE*, note) **UE**(<u>codeUE</u>, niveau, titre)
  - 100 Etudiants, 600 notes, 30 UE
  - $card(Etu \bowtie_{nomE} Note) = card(Note) = 600$
  - $card(Note \bowtie_{codeE} UE) = card(Note) = 600$
  - $card(Etu \bowtie_{nomE} Note \bowtie_{codeE} UE) = card(Note) = 600$

Cardinalité d'une jointure entre deux clés étrangères

- Cas d'une jointure $R \bowtie_A S$ entre 2 clés étrangères
  - Entité(A, ...), R(..., A*, ...) , S(..., A*, ...)
  - dom(R.A) ⊆ dom(Entité.A) donc D(Entité, A) ≥ D(R, A)
  - dom(S.A) ⊆ dom(Entité.A) donc D(Entité, A) ≥ D(S, A)

- $card(R \bowtie_A S) = card(R) * card(S) / D(Entité, A)$

- Exple :
  - **Etu**(nomE, age) et D(Etu, nomE) = **100**
  - **InscritSport**(sport, nomE*, annee)
    - D(InscritSport, nomE) = 20
  - **Résa** (codeLivre*, nomE*, date, bibliothèque, durée)
    - D(Resa, NumE) = 50
  - 100 Etudiants, 40 InscritSport, 200 Résa de livres
  - $card(InscritSport \bowtie_{nomE} Résa) = 40 * 200 / \mathbf{100} = 80$

Cardinalité d'une jointure entre 2 sélections

- Jointure entre deux expressions contenant des sélections
- Réécrire la jointure :
  - Exprimer la jointure avant la sélection
  - Commutativité de la composition des opérations :
    - jointure(sélection1(x), sélection2(y)) = sélection1(sélection2(jointure(x,y)))

$$card(\ \sigma_{p1}(R) \quad \bowtie_A \quad \sigma_{p2}(S)\ ) =$$

$$SF(p1) * SF(p2) * card(R \bowtie_A S)$$

Espace de recherche

- Caractérisé par les plans “équivalents” pour une même requête
  - ceux qui donnent le même résultat
  - générés en appliquant les règles de transformation vues précédement
- Le coût de chaque plan est en général différent
- L'ordre des jointures est important

Règles de transformation

- Commutativité des opérations binaires
  - $R \times S \equiv S \times R$
  - $R \bowtie S \equiv S \bowtie R$
  - $R \cup S \equiv S \cup R$
- Associativité des opérations binaires
  - $(R \times S) \times T \equiv R \times (S \times T)$
  - $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
- Idempotence des opérations unaires
  - $\Pi_{A'}(\Pi_{A''}(R)) \equiv \Pi_{A'}(R)$
  - $\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) \equiv \sigma_{p_1(A_1) \wedge p_2(A_2)}(R)$
    - où $R[A]$ et $A' \subseteq A$, $A'' \subseteq A$ et $A' \subseteq A''$

Règles de transformation

- Commutativité de la sélection et de la projection (si proj. des attr. sél.)
- Commutativité de la sélection avec les opérations binaires

$$\sigma_{p(A)}(R \times S) \equiv (\sigma_{p(A)}(R)) \times S$$

$$\sigma_{p(A_i)}(R \bowtie_{(A_j,B_k)} S) \equiv (\sigma_{p(A_i)}(R)) \bowtie_{(A_j,B_k)} S$$

$$\sigma_{p(A_i)}(R \cup T) \equiv \sigma_{p(A_i)}(R) \cup \sigma_{p(A_i)}(T)$$

où A_i appartient à R et T

- Commutativité de la projection avec les opérations binaires

$$\Pi_C(R \times S) \equiv \Pi_{A'}(R) \times \Pi_{B'}(S)$$

$$\Pi_C(R \bowtie_{(A_j,B_k)} S) \equiv \Pi_{A'}(R) \bowtie_{(A_j,B_k)} \Pi_{B'}(S)$$

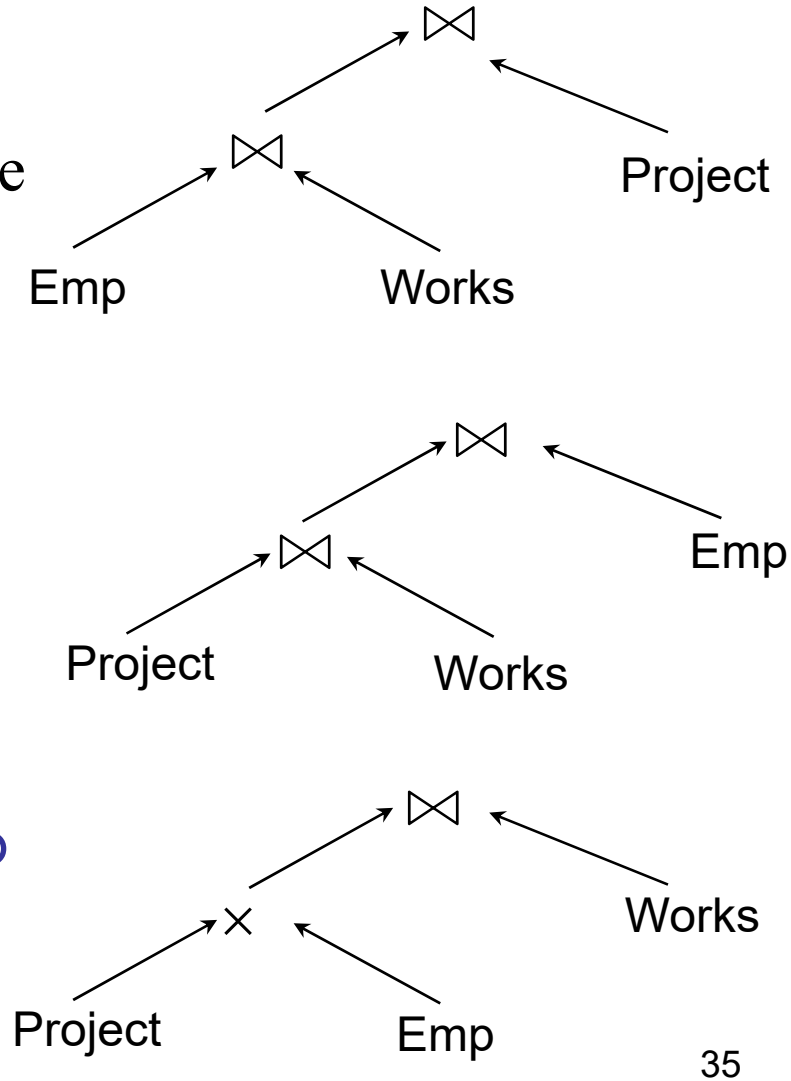
$$\Pi_C(R \cup S) \equiv \Pi_C(R) \cup \Pi_C(S)$$

où $R[A]$ et $S[B]$; $C = A' \cup B'$ où $A' \subseteq A$, $B' \subseteq B$, $A_j \subseteq A'$, $B_k \subseteq B'$

Ordre des jointures

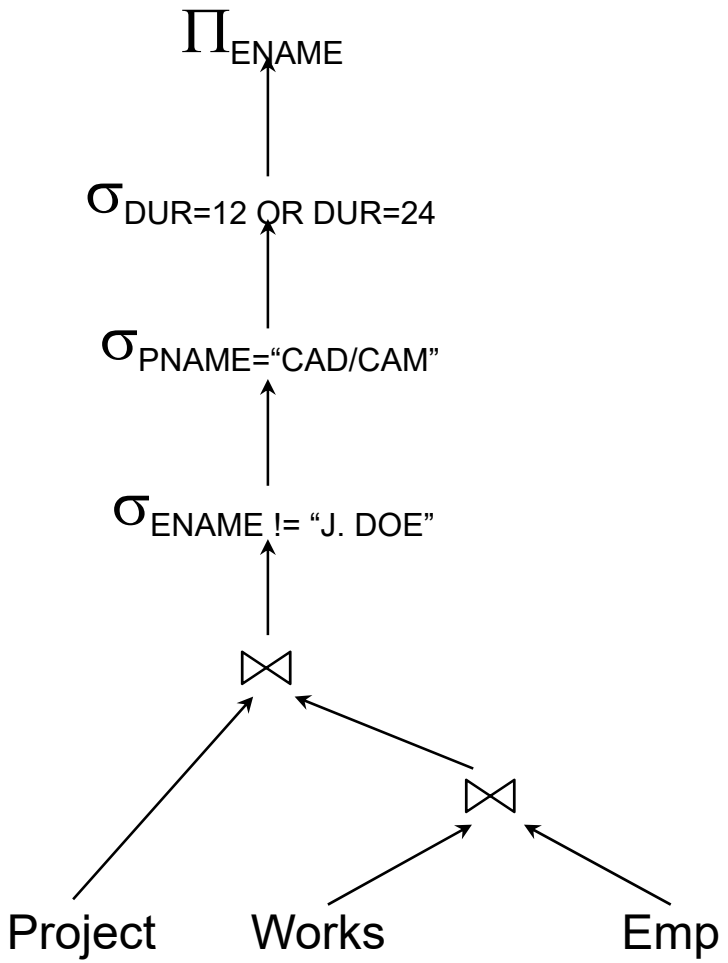
- Avec *N* relations, il y a O(***N!***) ordres de jointures équivalents qui peuvent être obtenus en appliquant les règles de *commutativité* et d'*associativité*

```
SELECT    Ename,Resp
FROM      Emp, Works,Project
WHERE     Emp.Eno=Works.Eno
AND       Works.PNO=Project.PNO
```

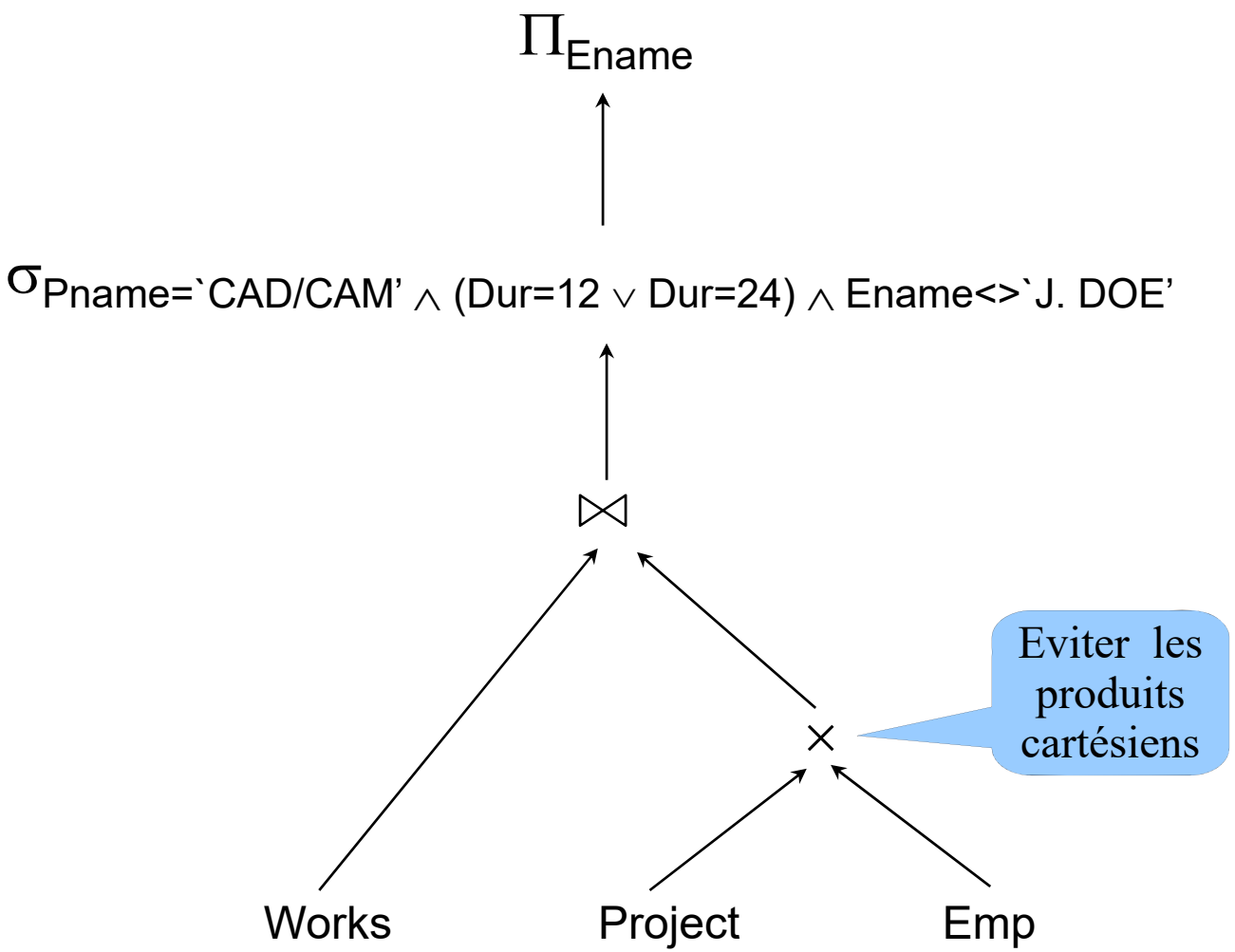


Exemple

```
SELECT Ename
FROM Project p, Works w,
     Emp e
WHERE w.Eno=e.Eno
AND   w.Pno=p.Pno
AND   Ename<>`J. Doe'
AND   p.Pname=`CAD/CAM'
AND   (Dur=12 OR Dur=24)
```

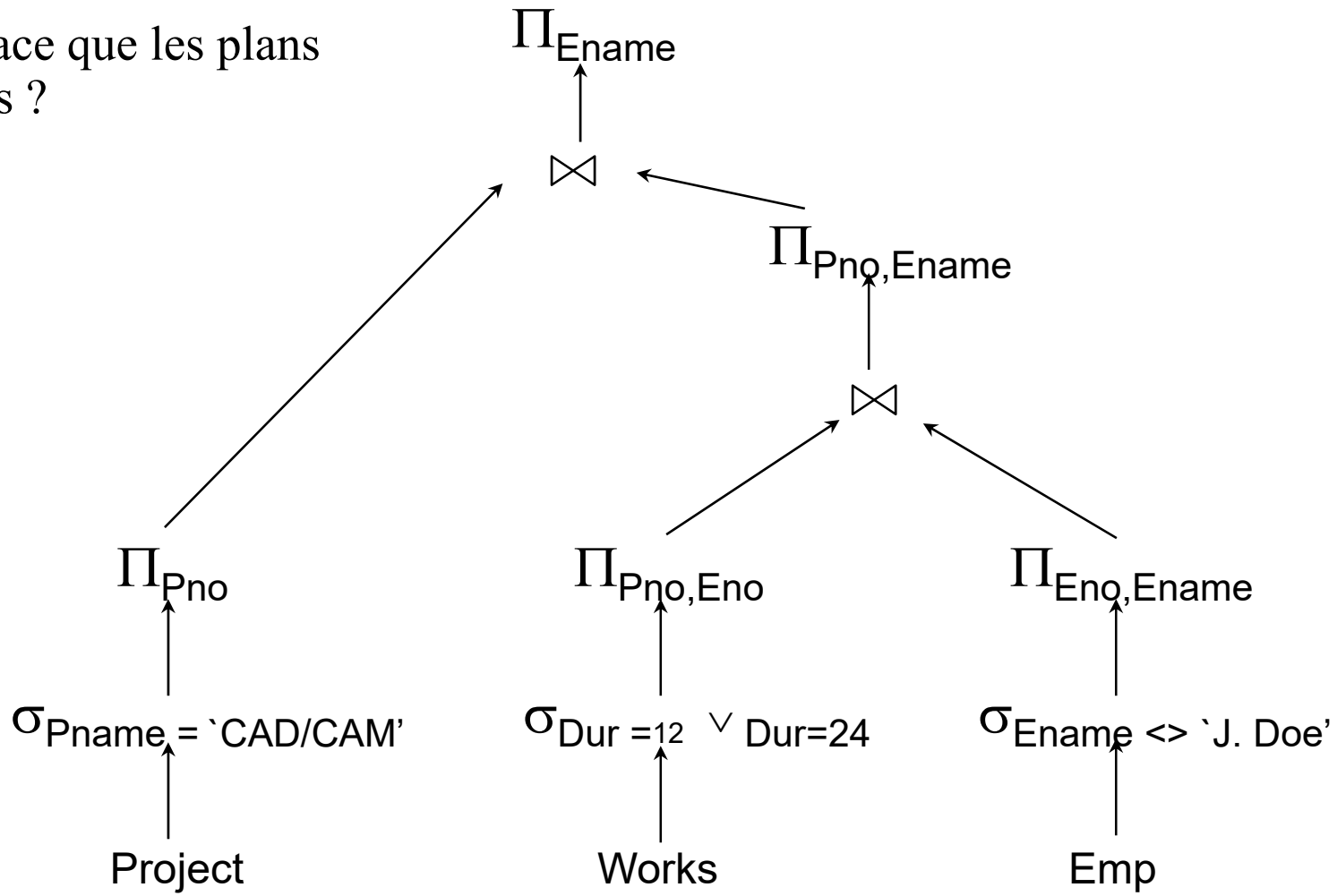


Plan équivalent

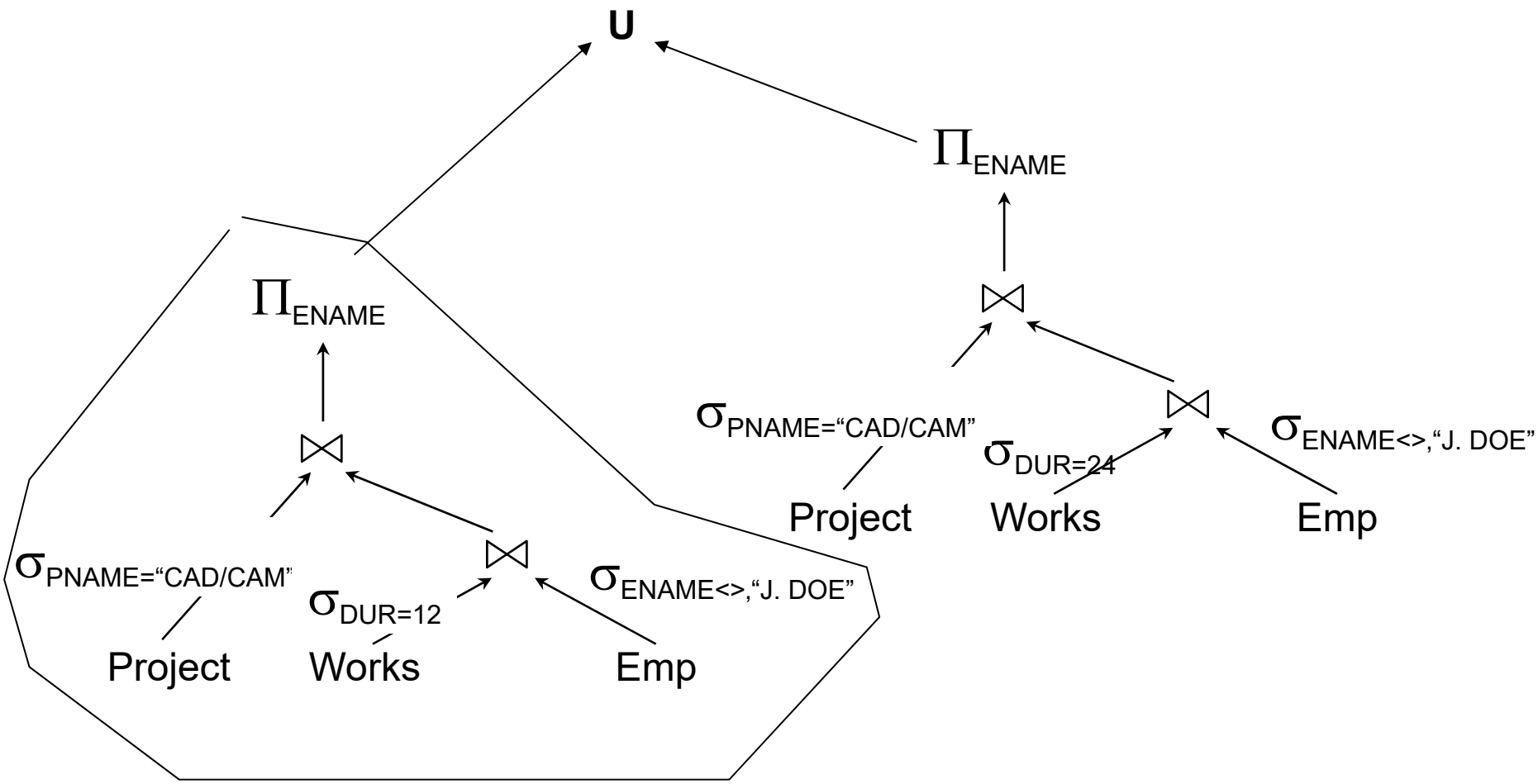


Autre plan équivalent

Plus efficace que les plans précédents ?



Encore un autre plan équivalent

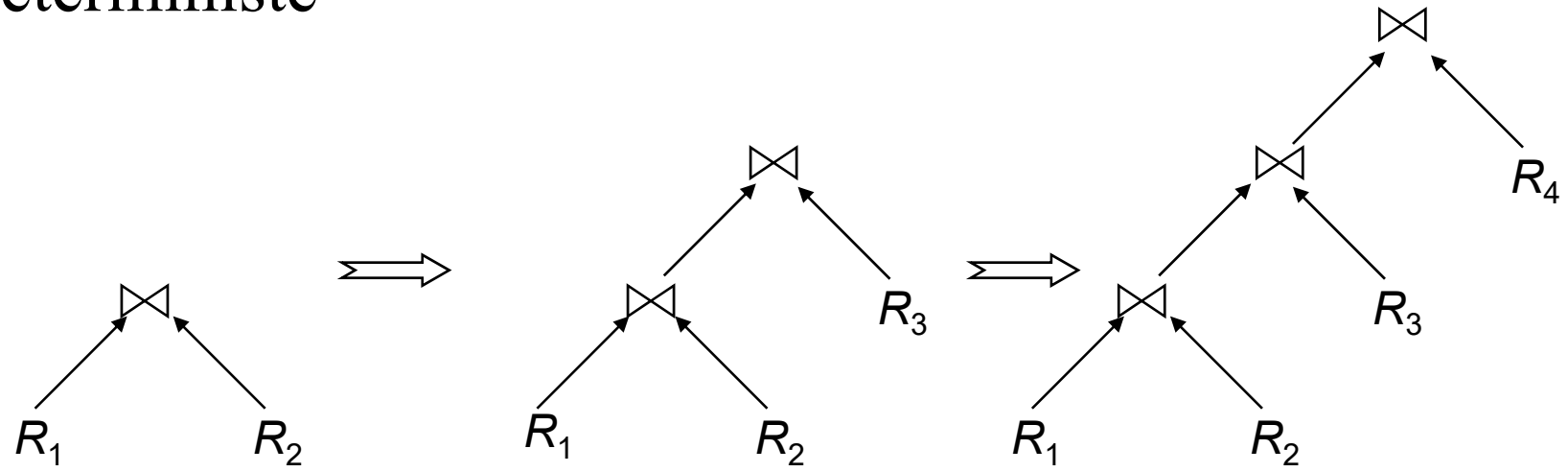


Stratégie de recherche

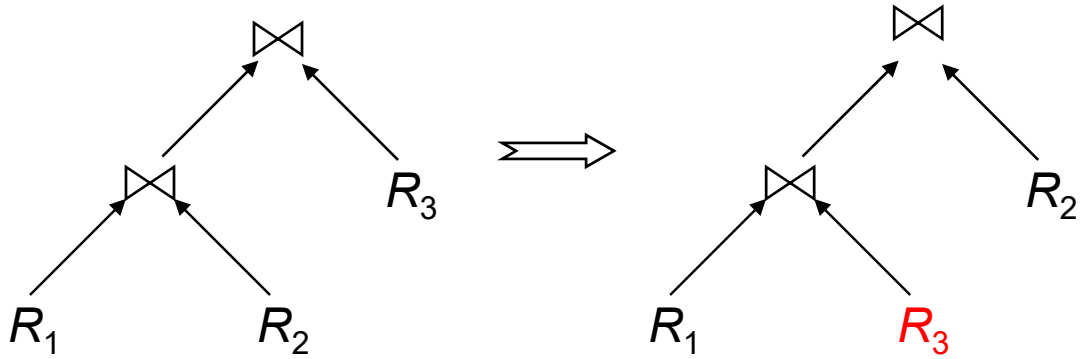
- Il est en général trop coûteux de faire une recherche exhaustive
- Déterministe
  - part des relations de base et construit les plans en ajoutant une relation à chaque étape
  - programmation dynamique: largeur-d'abord
  - excellent jusqu'à 5-6 relations
- Aléatoire
  - recherche l'optimalité autour d'un point de départ particulier
  - réduit le temps d'optimisation (au profit du temps d'exécution)
  - meilleur avec > 5-6 relations
    - **recuit simulé (simulated annealing)**
    - **amélioration itérative (iterative improvement)**

Stratégies de recherche

- Déterministe



- Aléatoire



Stratégie déterministe pour obtenir un seul plan

- Les relations $R = \{R_1, \ldots, R_n\}$
- P_1 : 1ère jointure, énumérer les paires

  $P_1 = \{(Ri \bowtie Rj) \mid Ri, Rj \in R \text{ et } i \neq j\}$

  $Pmin_1$ = la paire de coût min = $\text{argmin}_{p \in P1}$ coût(p)
- P2 : jointure entre $Pmin_1$ et une autre relation

  P2 contient seulement n-2 plans

...

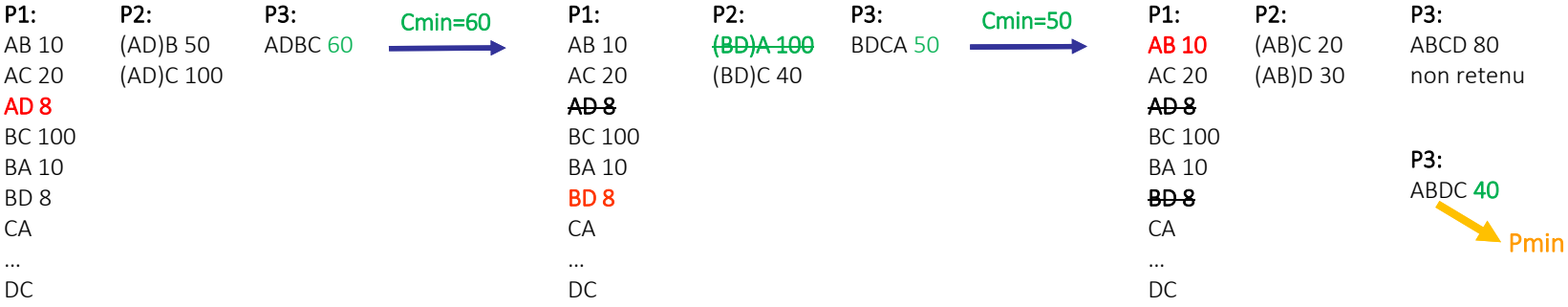
- $P_{n-1} = \{(Pmin_{n-2} \bowtie Ri) \mid Ri \in R/\text{relations de } Pmin_{n-2}\}$

  $\rightarrow P_{n-1}$ contient un seul plan

Exploration plus large avec élagage

- Hypothèse : $P = P' \bowtie R$ coût(P) > coût(P')
- Obtenir un premier plan : Pmin
  - calculer les ensembles de sous-plans P_1, …, P_{n-1}
  - ne pas se limiter au sous plan $Pmin_k$ de chaque ensemble P_k
  - mais explorer d'autres sous plans P' dont le coût est inférieur à Pmin
- Méthode récursive :
  - Considérer, dans l'ordre croissant de coût, les autres plans P' de P_{n-2} et qui ont un coût < Cmin
    - compléter P' pour obtenir une solution: P' remplace le meilleur plan si son coût < Cmin
    - condition d'arrêt : durée, nombre de plans

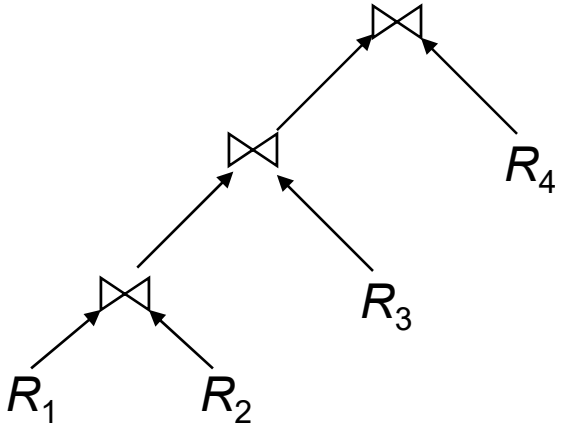
Exemple pour joindre les relations ABCD



Forme des arbres explorés

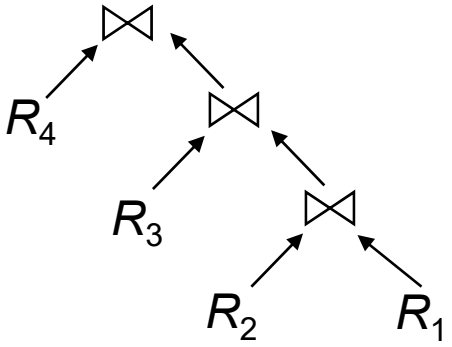
- Limiter l'espace de recherche
  - heuristiques
    - par ex. appliquer les opérations unaires avant les autres
    - Ne marche pas toujours (perte d'index, d'ordre)
  - limiter la forme des arbres de jointure

Arbre linéaire à gauche



= ((R1 J R2) J R3) J R4

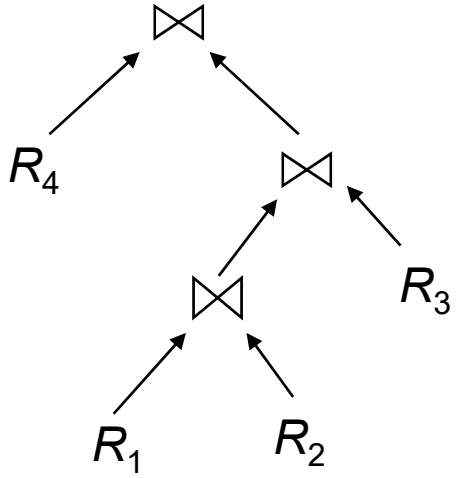
Arbre linéaire à droite



= R4 J (R3 J (R2 J R1))

Arbres zig-zag et *bushy*

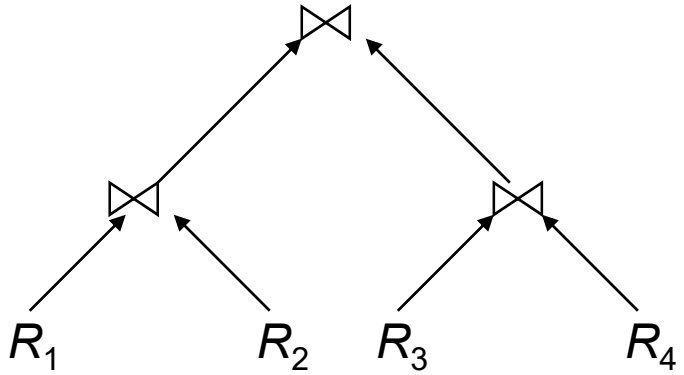
Arbres zig-zag



= R4 J ((R1 J R2) J R3)

autre arbre zig-zag:
(R3 J (R1 J R2)) J R4

Arbre touffu (bushy tree)



Génération de plan physique

- Sélection :
  - Commencer par les conditions d'égalité avec un index sur l'attribut
  - Filtrer sur cet ensemble de n-uplets ceux qui correspondent aux autres conditions
- Jointure
  - Utilisation des index, des relations déjà triées sur l'attribut de jointure, présence de plusieurs jointures sur le même attribut
- Pipelines ou matérialisation

Conclusion

- Point fondamental dans les SGBD
- Importance des métadonnées, des statistiques sur les relations et les index, du choix des structures d'accès.
- L'administrateur de bases de données peut améliorer les performances en créant de nouveaux index, en réglant certains paramètres de l'optimiseur de requêtes (voir TP)