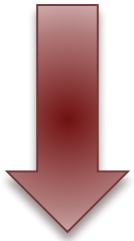
Évaluation de requêtes centralisées et distribuées

Rules Based Optimisation

Requête (SQL)

Traduction

Requête en algèbre



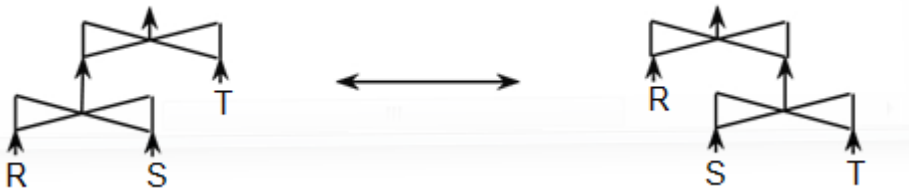
Transformation en fonction des propriétés des opérateurs

Plan optimal

Propriétés des opérateurs

- **Commutativité pour jointure et produit**

- **Associativité pour jointure et produit**



- Il existe N!/2 arbres de jointure de N relations.

Propriétés des opérations (suite)

- Groupage des projections
  - $\pi A1, \ldots, An\ (\pi B1, \ldots, Bm(E)) = \pi A1, \ldots, An\ (E)$ --- {A1, …, An} inclus dans {B1, …, Bm}
- Groupage des sélection
  - $\sigma_{F1}\ (\sigma_{F2}\ (E)) = \sigma_{F1 \wedge F2}(E)$
- Exemple : $\sigma_{age>20}\ (\sigma_{genre='F'}\ (\text{Étudiant})) = \sigma_{(age>20)\ \wedge\ (genre\ ='F')}(\text{Étudiant})$
- Inversion σ et π
  - $\pi_{A1, \ldots, An}(\sigma_F(E)) = \sigma_F(\pi_{A1, \ldots, An}(E))$ si F porte sur A1, …, An
  - $\pi_{A1, \ldots, An}(\sigma_F(E)) = \pi_{A1, \ldots, An}\ (\sigma_F(\pi_{A1, \ldots, An, B1, \ldots, Bn})\ (E))$ si F porte sur B1, …, Bm qui ne sont pas parmi A1, …, An

Propriétés des opérations (suite)

- Inversion Sélection-Produit
  - σ_F **(E1 * E2) =** σ_F **(E1) * E** si F ne porte que sur les attributs de E1
  - σ_F **(E1 * E2) =** σ_{F1} **(E1) *** σ_{F2} **(E2)** Si F = F1 ∧ F2 et Fi ne porte que sur les attributs de Ei
  - σ_F **(E1 * E2) =** σ_{F2} **(**σ_{F1} **(E1) * E2)** si F1 porte sur les attributs de E1 et F2 sur ceux de E1 et E2
  - Exemple : $\sigma_{Genre='F'}$(Étudiant * Cours) = $\sigma_{Genre='F'}$(Étudiant) * Cours
- Inversion Sélection-Union
  - $\sigma_F (E_1 \cup E_2) = \sigma_F(E_1) \cup \sigma_F(E_2)$
- Inversion Sélection-Diffétrence
  - $\sigma_F (E_1 - E_2) = \sigma_F(E_1) - \sigma_F(E_2)$
- Inversion Projection-Produit
- Inversion Projection-Union

Principes d'optimisation des expressions algébriques

1. Exécuter les sélections aussitôt que possible
2. Réduire la taille des relations à manipuler
    - **Combiner les sélections avec un produit cartésien pour aboutir à une jointure**
3. **Combiner des séquences d'opérations unaires (σ,Π)**
4. Mémoriser les sous-expression commune

Exécuter les sélections aussitôt que possible

- Appliquer ces règles (remplacer membre gauche par droit)

1. $\sigma_F (E1 \cup E2) = \sigma_F(E1) \cup \sigma_F(E2)$
2. $\sigma_F (E1 - E2) = \sigma_F(E1) - \sigma_F(E2)$
3. $\sigma_F (E1 * E2) = \sigma_F (E1) * E2$ si F ne porte que sur les attributs de E
4. $\sigma_F (E1 * E2) = \sigma_{F1} (E1) * \sigma_{F2} (E2)$ si F = F1 ∧ F2 et Fi ne porte que sur les attributs de Ei
5. $\sigma_F (E1 * E2) = \sigma_{F2} (\sigma_{F1} (E1) * E2)$ si F porte sur les attributs de E1 et F2 sur ceux de E1 et E2

Combiner les sélection avec un produit cartésien pour aboutir à une jointure

- Transformation si possible du produit cartésien en jointure:
  - Si R * S est l'argument d 'une sélection σ_F alors si F est une comparaison entre attributs de R et de S, transformer R * S en jointure : $\sigma_F (R * S) = (R \infty_F) S$
- Attention : si F est une expression ne portant que sur les attributs de R:
  - $\sigma_F (R * S) = \sigma_F (R) * S$

Sous-expressions communes à une expression

- Si une expression contient plusieurs fois la même sous-expression
- Si cette sous-expression peut être lue en moins de temps qu'il faut pour la calculer
- Matérialiser la sous-expression commune
- Principe de vues matérialisées

Un algorithme d'optimisation

1. Séparer chaque sélection $\sigma_{F1 \wedge \ldots \wedge Fn}(E)$ en une cascade $\sigma_{F1}(\sigma_{F1} \ \ldots (\sigma_{Fn}(E)))$
2. Descendre chaque sélection le plus bas possible dans l'arbre algébrique
3. Descendre chaque projection aussi bas possible dans l'arbre algébrique
4. Combiner des cascades de sélection et de projection dans une sélection seule, une projection seule, ou une sélection suivie par une projection
5. Regrouper les nœuds intérieurs de l'arbre autour des opérateurs binaires (*, -, U). Chaque opérateur binaire crée un groupe
6. Produire un programme comportant une étape pour chaque groupe, à évaluer dans n'importe quel ordre mais tant qu'aucun groupe ne soit évalué avant ses groupes descendants.

Quelques problèmes

- La restructuration d'un arbre sous-entend souvent la permutation des opérateurs binaires
  - Cette optimisation ignore la taille de chaque table, les facteurs de sélectivité, etc.
  - Aucune estimation de résultats intermédiaires
- Exemple:

- (Ouvrier ∞ Usine) ∞ Contrat
- Si cardinalité (Usine) <<<Cardinalité(Contrat) << Cardinalité(Ouvrier) alors l'expression suivante a des chances d'être plus performante:
- (Usine ∞ Contrat) ∞ Ouvrier
- → Optimisation basée sur les modèles de coût

Méthodes d’optimisation basées sur les modèles de coûts

Notions utiles

- Les tables relationnelles sont stockées physiquement dans des fichiers sur disque
- Lorsqu'on veut accéder aux données, on doit transférer le contenu pertinent des fichiers dans la mémoire
  - Fichier = séquence de tuples
  - Bloc (page) = unité de transfert de données entre disque et mémoire
  - Facteur de blocage = nombre de tuples d'une relation qui «tiennent» dans un bloc
  - Coût de lecture (ou écriture) d'une relation = nombre de blocs à lire du disque vers la mémoire (ou à écrire de la mémoire vers le disque)

Estimation du coût des opérations physiques

- **TempsES : temps d'accès à mémoire secondaire (MS)**
- **TempsUCT : Souvent négligeable**
- **TailleMC : espace mémoire centrale**
- **TailleMS : espace mémoire secondaire**

Modèle du coût d'une entrée- sortie en mémoire secondaire

Paramètre	Signification
TempsESDisque(n)	Temps total de transfert (lecture ou écriture) de n octets du disque
TempsTrans(n)	Temps de transfert des n octets sans repositionnement
TempsPosDébut	Temps de positionnement au premier octet à transférer (ex : 10 ms)
TempsRotation	Délai de rotation (ex : 4 ms)
TempsDépBras	Temps de déplacement du bras (ex : 6 ms)
TauxTransVrac	Taux de transfert en vrac (ex : 2MB/sec)
TempsESBloc	Temps de transfert d'un bloc (ex : 11 ms)
TempsTrans	Temps de transfert d'un bloc sans repositionnement (ex : 1 ms)
TailleBloc	Taille d'un bloc (ex : 2K octets)

Statistiques au sujet des tables

Statistique	Signification
||T||	Nombre de lignes de la table T
TailleLigne	La taille d'une ligne de la table T
FBT	Facteur de blocage moyen de T
NDIST	Nombre de valeurs distinctes de la colonne pour la table T
Min_T(colonne)	Valeur minimum de la colonne de T
Max_T(colonne)	Valeur maximum de la colonne de T
|T|	Nombre de pages de la table T

Facteur de sélectivité

- ||(σ(R))|| = SF * ||R||
  - SF (A = valeur) = 1 / NDIST(A)
  - SF(A > valeur) = (max(A) - valeur) / (max(A) - min(A))
  - SF(A < valeur) = (valeur - min(A)) / (max(A) - min(A))
  - SF(A IN liste valeurs) = (1/NDIST(A)) * CARD(liste valeurs)
  - SF(P et Q) = SF(P) * SF(Q)
  - SF(P ou Q) = SF(P) + SF(Q) - SF(P) * SF(Q)
  - SF(not P) = 1 - SF(P)
- Le coût dépend de l'algorithme (index, hachage ou balayage).

Facteur de sélectivité

- **La taille d'une jointure est estimée par la formule suivante:**
- **||R1 ∞ R2|| = Sel * ||R1|| * ||R2||;**
- **avec sel: la sélectivité de la jointure**

Exemples

```
SELECT *
FROM R1, R2
```

SF =1

```
SELECT *
FROM R1
WHERE A = valeur
```

- SF = 1/NDIST(A) avec un modèle uniforme

- **Algorithmes Généraux**
- **pour les opérateurs relationnels**

La sélection

- Sélection sans index

- SELECT NSS, Nom FROM Ouvrier WHERE Salaire > 15000

- Plan d'exécution

  - pour chaque tuple t de ouvrier faire -- accès séquentiel

  - si **t.salaire > 15000** alors **réponse = réponse** $\cup$ πNSS, Nom (Ouvrier)

Algorithmes de jointure

- Jointure sans index
  - **Jointure par boucles imbriquées (nested loop)**
  - **Jointure par tri-fusion (sort-join)**
  - **Jointure par hachage (hash-join)**
- Jointure avec index
  - **Jointure avec boucles indexées**

Jointure par boucles imbriquées (JBI)

- $R \infty_F S$
- Principe:
  - La première table est lue séquentiellement et de préférence stockée entièrement en mémoire
  - Pour chaque tuple de R, il y a une comparaison avec les tuples de S

Jointure par boucles imbriquées

POUR chaque ligne lR de R
POUR chaque ligne lS de S
SI θ sur lR et lS est satisfait
Produire la ligne concaténée à partir de lR et lS
FINSI
FINPOUR
FINPOUR

- + Algorithme simple
- Complexité: (||R|| + (||R||*||S||)
- Si la table extérieure tient en mémoire: une seule lecture des deux tables suffit.

Jointure par tri fusion

- **Plus efficace que les boucles imbriquées pour les grosses tables**
- **Principe:**
  - **Trier les deux tables sur les colonnes de jointure**
  - **Effectuer la fusion**
  - **Complexité: Le tri qui coûte cher**

Jointure par tri fusion

Trier R et S par tri externe et réécrire dans des fichiers temporaires

Lire groupe de lignes GR(cR) de R pour la première valeur cR de clé de jointure

Lire groupe de lignes GS(cS) de S pour la première valeur cS de clé de jointure

TANT QUE il reste des lignes de R et S à traiter

SI cR = cS

Produire les lignes concaténées pour chacune des combinaisons de lignes de GR(cR) et GS(cS);

Lire les groupes suivants GR(cR) de R et GS(cS) de S;

SINON

SI cR < cS

Lire le groupe suivant GR(cR) de R

SINON

SI cR > cS

Lire le groupe GS(cS) suivant dans S

FINSI

FINSI

FINSI

FIN TANT QUE

Jointure par tri fusion

- **Très efficace quand une des deux tables est petite (1, 2, 3 fois la taille de la mémoire)**
- **Algorithme en option dans Oracle**

Principe de la jointure par hachage

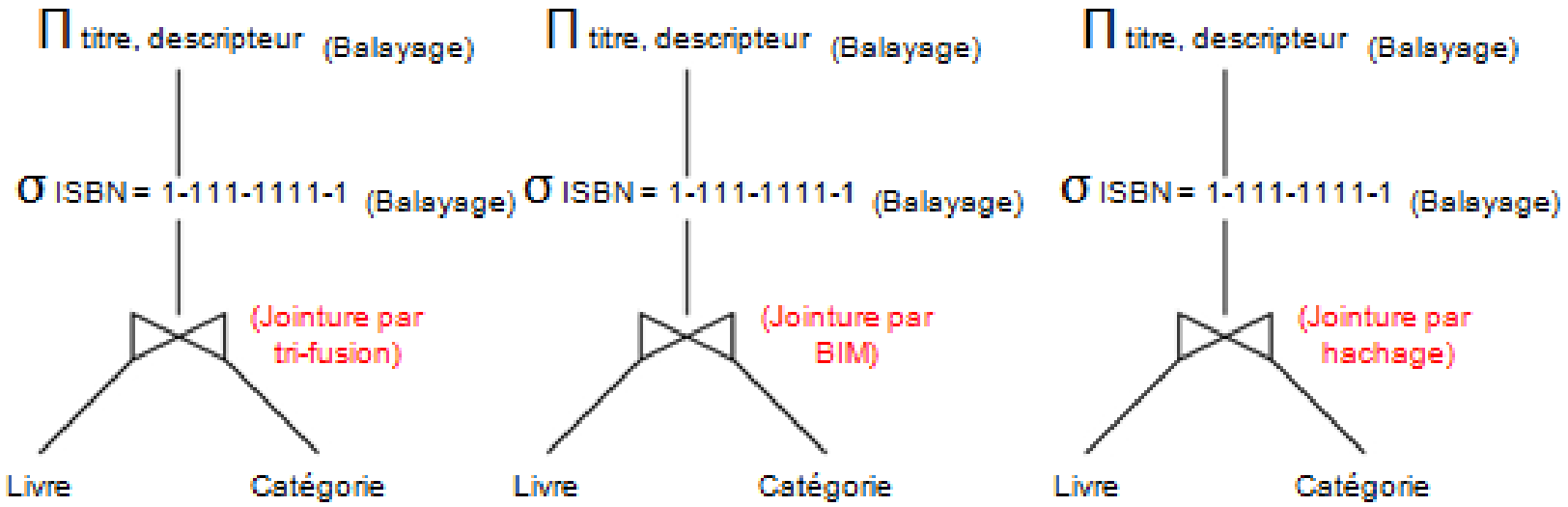
1. Hacher la plus petite des deux tables en n fragments
2. Hacher la seconde table, avec la même fonction, en n autres fragments
3. Réunir fragments par paire, et faire la jointure entre les fragments

Comment se fait l'Optimisation

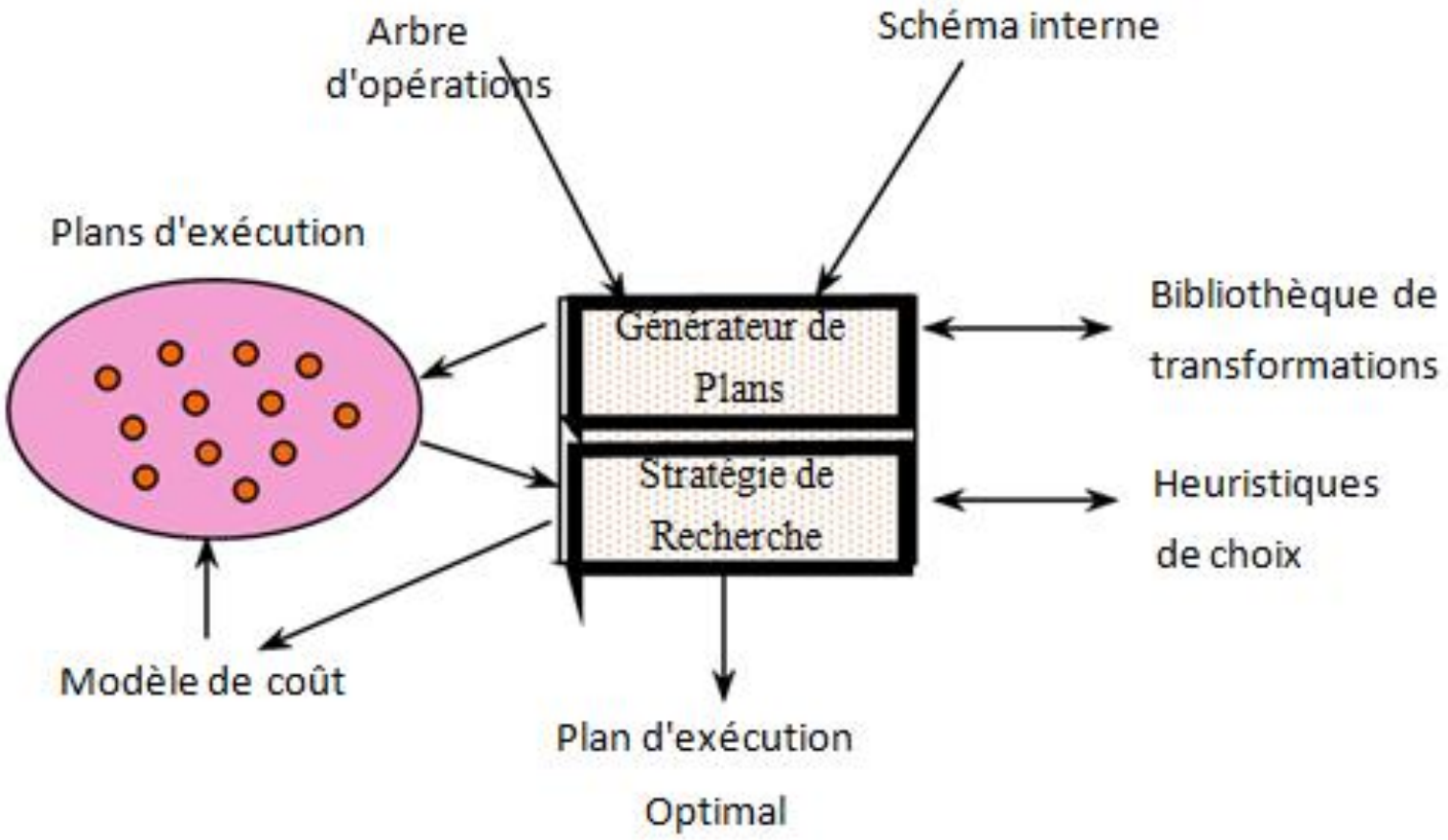
- Chercher le meilleur plan d 'exécution?
  - coût excessif
- Solution approchée à un coût raisonnable
  - Générer les alternatives
    - heuristiques
- Choisir la meilleure estimation approximative du coût

Plusieurs plans d'exécution pour un arbre algébrique

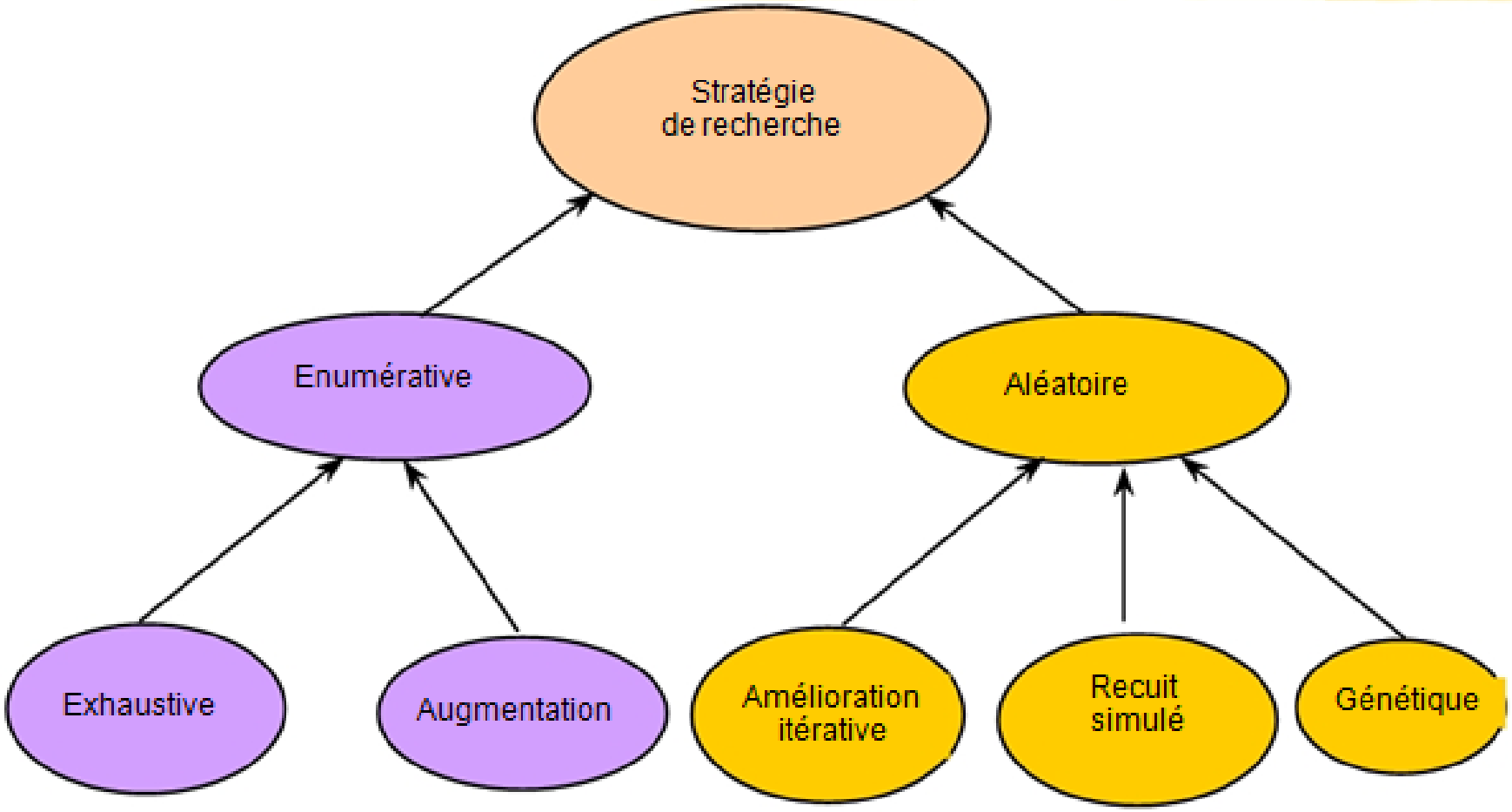
- Pour chaque opération logique
  - plusieurs choix d 'opérations physiques



Choix du meilleur plan



Différentes Stratégies



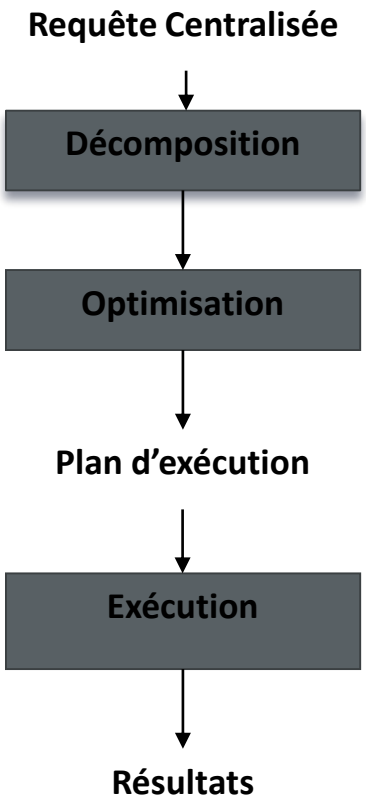
Optimisation basée sur modèle de coût

- Soit Q une requête à optimiser
- Procédure

1. Énumérer tous les plans $\{P_1, \ldots, P_m\}$ pour chaque requête (notons que chaque requête possède un ensemble d'opérations $O_1, \ldots, O_k$)
2. Pour chaque plan P_j
   - Pour chaque opération O_i du plan P_j, énumérer les chemins d'accès
   - Sélectionner le chemin ayant le coût le moins élevé

   $Coût(P_i) = \Sigma_{(l=1, k)} \min(O_l)$
3. Coût (Q): $\Sigma_{(h=1, m)}$ Coût(P_h)

Traitement des requêtes centralisée



Évaluation d'une requête centralisée

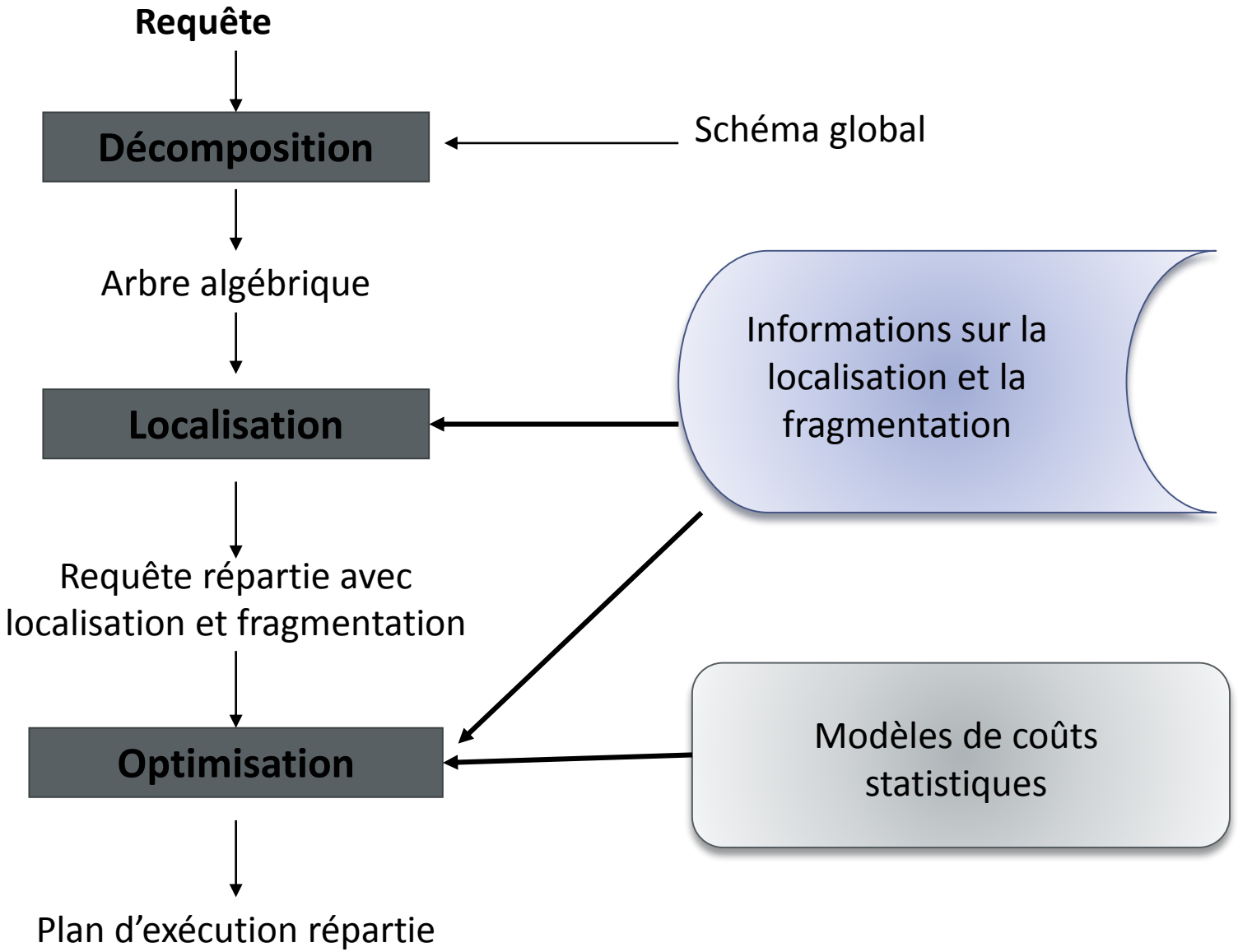
Décomposition *:* durant cette étape, la requête SQL est transformée en arbre algébrique.

Optimisation : des optimisations sont appliquées sur l' arbre issu de la phase précédente, afin de déterminer l'ordonnancement optimal des opérations relationnelles, ainsi que les algorithmes d'accès aux données. L'optimisation utilise :

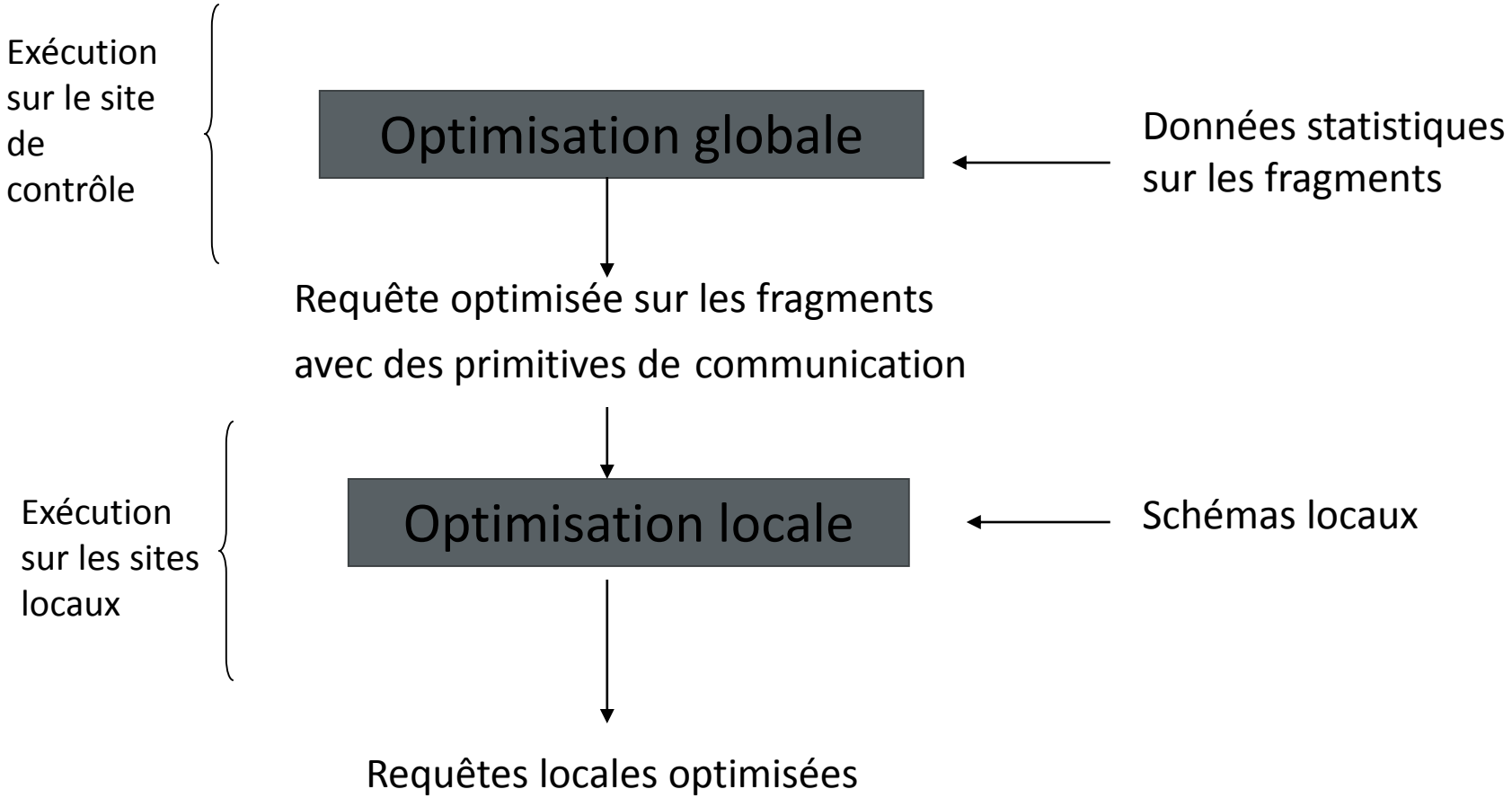
- Les propriétés des opérateurs algébriques (Associativité, Commutativité, Distributivité)
- Des heuristiques : en évaluant en premier les opérations les moins coûteuses, par exemple en faisant descendre *les sélections,* ce qui réduit le nombre de tuples, ainsi que *les Projections* qui réduit le nombre de colonnes.

Exécution : c'est l'étape finale, qui consiste dans l'exécution de la requête optimisée afin d'élaborer son résultat.

Traitement des requêtes réparties



Optimisation de requêtes distribuées



Optimisation de requêtes distribuées

- **Décomposition de requête**: cette couche prend la requête exprimée en termes de relations globales et effectue une première optimisation partielle à l'aide d'heuristiques sur l'ordonnancement des opérateurs

- **Localisation de données:** cette couche prend en compte la répartition des données sur le système. On remplace les feuilles de l'arbre d'algèbre relationnelle par leurs algorithmes de reconstruction

- **Optimisation globale**: cette couche prend en compte les informations statistiques pour trouver un plan d'exécution proche de l'optimum, basé sur les fragments

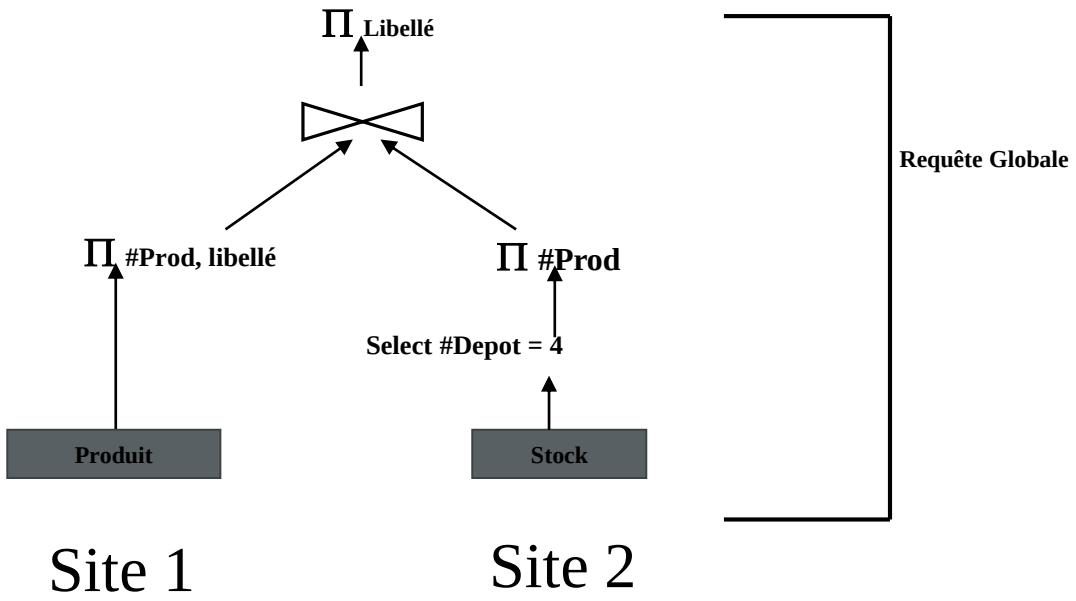
- **Optimisation locale**: cette couche s'exécute sur chacun des sites locaux impliqués dans la requête. Chaque SGBD local effectue ses propres optimisations à l'aide des heuristiques classiques d'optimisation

La décomposition

La requête globale est décomposée en sous-requêtes locales.

Relation : *Produit* (Prod, Libellé, Pu…,) ; *Stock* (Dépôt,…)

La requête globale *Req* = Quels sont les produits du dépôt n°4 ?



Optimisation de requêtes: la localisation

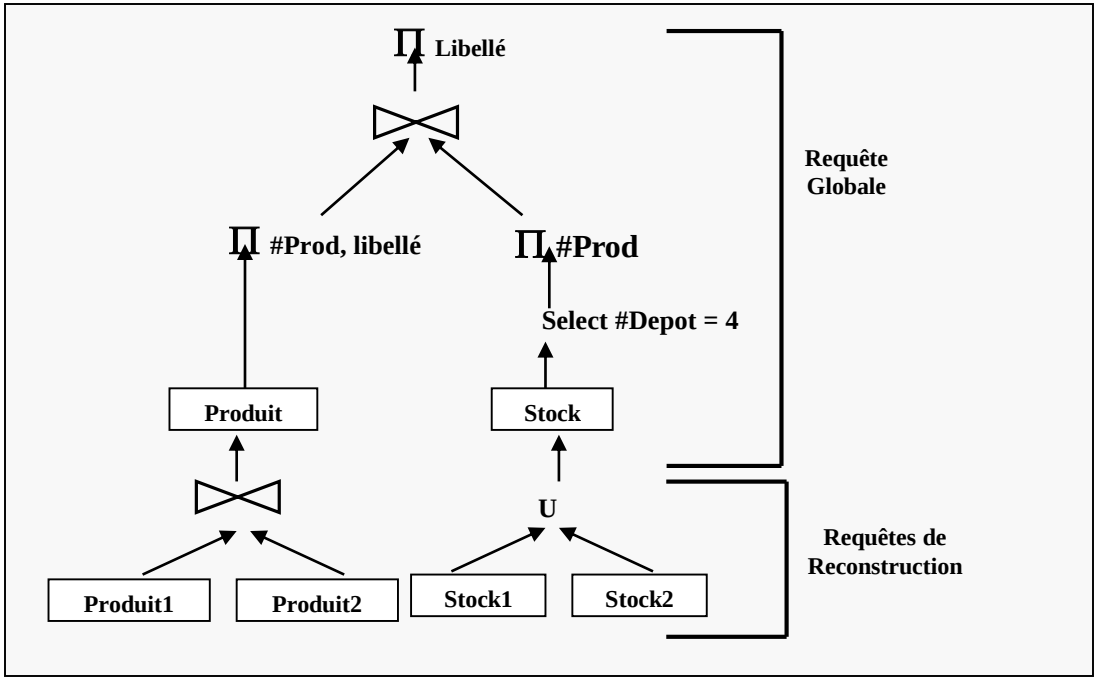
❑**La localisation** d'une requête distribuée procède en deux étapes :

- ❑Génération de requête canonique
- ❑Simplification (ou réduction)

❑ ***Génération de requête canonique***

- ❑ la requête canonique est obtenue en remplaçant chaque relation de la requête distribuée globale par la requête de reconstruction correspondante : c'est-à-dire il est remplacé les relations globales des feuilles de l'arbre par leurs fragments constitutifs

Exemple



Site 1

Site 2

Produit1= **Proj** #prod,libellé (Produit)

Produit2 = **Proj** #prod,Pu (Produit)

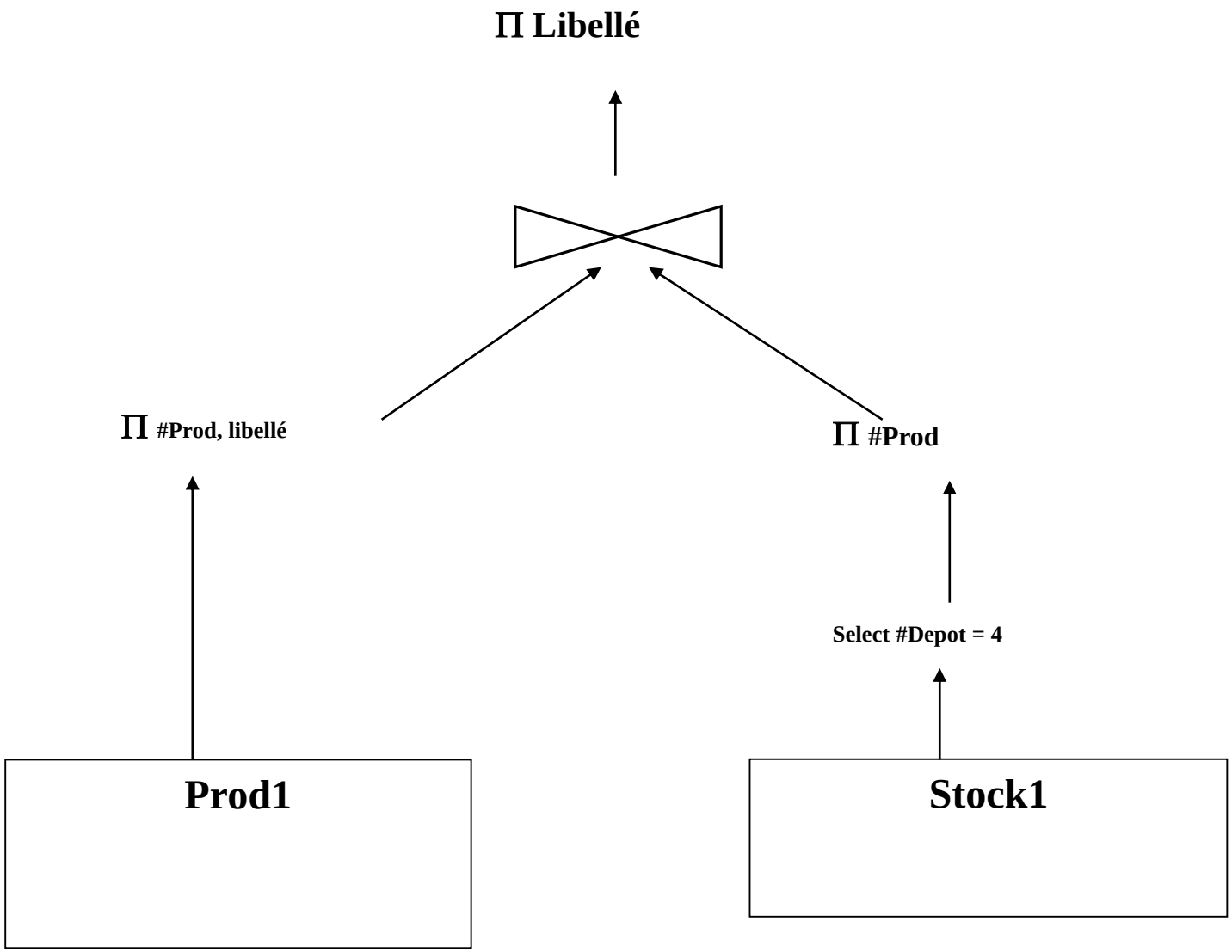
Stock1 = **SEL** # prod,Dep=4 (Stock)

Stock2 = **SEL** # prod,Dep <> 4 (Stock)

Optimisation des requêtes réparties: la simplification

- ***La simplification*** *:* elle est basée sur des règles de simplifications, qui permettent de déterminer les opérations inutiles dans une requête canonique. Les opérations inutiles sont celles qui produisent un résultat vide, ou identique à l'opérande. Parmi ces règles, on cite :
  - Une opération de restriction sur un fragment horizontal, dont le prédicat est en contradiction avec le prédicat de fragmentation, produit un résultat vide. *Exemple: la restriction à dépôt=4 au fragment défini sur dépôt<>4 (Stock2)*
  - Éliminer les conditions de sélections *inutiles,* quand la condition est identique à celle de la fragmentation. *Exemple: sélection à dépôt=4 du fragment défini sur dépôt=4 (Stock 1)*
  - Une opération de projection sur un fragment vertical, dont tous les attributs projetés- à l'exception, de l'attribut commun de reconstruction- n'appartiennent pas au fragment, donne un résultat vide . *Exemple: projection sur numprod,libellé du fragment Produit1*

Arbre final



Traitement des requêtes réparties

Le rôle de l'optimisation, est de déterminer une stratégie d'exécution de la requête distribuée qui minimise une fonction de coût, en évitant ainsi des stratégies qui conduisent à de mauvaises performances.

Les fonctions de coût à minimiser sont généralement les suivantes

- ***Le temps total d'exécution***
  - Dans le cas d'un SGBD centralisé, ce temps est la somme, du temps des entrées-sorties (accès disque) et du temps du calcul en unité centrale. Dans le cas distribué, on ajoute le temps de communication nécessaire, à l'échange des données entre différents sites participants à la requête distribuée.
- ***Le temps de réponse***
  - Il prend en compte les traitements exécutés en parallèle.

Objectifs et facteurs de l'optimisation

- Minimiser la fonction coût d'exécution : somme des temps d'exécution des sous requêtes sur chaque site
- Tenir compte :
  - Du parallélisme
  - Des coûts de transferts
  - Des profils des fragments
    - Taille, nombre de n-uplets
    - Taille du domaine des attributs…
  - De la taille des résultats intermédiaires
  - De la topologie du réseau

Traitement de requêtes réparties

L'exécution : le processus d'exécution de requêtes est aussi distribué, c'est-à-dire qu'il consistera en plusieurs processus d'exécution de requêtes locales et d'un processus d'exécution global.

- ***Le plan de l'exécution*** est l'ensemble des traitements locaux, ainsi que les opérations de communications des données intermédiaires, nécessaires pour les exécutions locales et la synchronisation globale.

Traitement de requêtes réparties

- Les décisions à prendre dans le plan d'exécution sont les suivantes
  - Ordre des jointures
  - stratégie de jointure
  - Sélection des copies (site le plus proche, le moine engorgé)
  - Choix des sites d'exécution (fonction des coûts de communication)
  - Choix des algorithmes d'accès répartis
  - Choix du mode de transfert (tuple/paquets)

GESTION DU CATALOGUE

- Dans un système distribué, le catalogue du système contiendra non seulement les données d'un catalogue classique (relations de base, vues, index, …) mais également toutes les informations de contrôle nécessaires au système pour assure la fragmentation , la duplication..

Quelques possibilités

1. **Centralisé**
   - l'ensemble du catalogue est mémorisé une seule fois, sur un site central unique.
- ***Cette approche viole l'objectif « pas de contrôle centralisé »***

2. **Duplication totale**
   - l'ensemble du catalogue est entièrement mémorisé sur chaque site.
- ***Perte importante d'autonomie***

GESTION DU CATALOGUE

- **3. Partitionné**
  - Chaque site gère son propre catalogue concernant les objets de ce site. L'ensemble du catalogue est l'union de tous les catalogues locaux et disjoints.
- ***Les opérations qui ne sont pas locales sont très coûteuses (trouver un objet distant nécessite d'accéder en moyenne à la moitié des sites)***
- **Combinaison de 1 et 3**
  - Chaque site gère son propre catalogue local, de plus un site central gère une copie unifiée de tous les catalogues locaux.
- *Plus efficace que 3 mais viole l'objectif « pas de contrôle centralisé ».*