Chapitre 6 : Optimisation de Requêtes

Plan

1. **INTRODUCTION**

2. **L'ANALYSE DE QUESTION**

3. **ORDONNANCEMENT DES OPERATEURS**
   a. **Règles de transformation des arbres**
   b. **Algorithme simple de génération d'un plan de requêtes**

4. **ESTIMATION DES COUTS D'EXECUTION**
   a. **Utilisation de données statistiques**
   b. **Utilisation de procédures d'implantation des opérateurs**

Chapitre 6 : OPTIMISATION DE REQUETES

1. INTRODUCTION

Un des objectifs des bases de données est de restituer l'information dans des délais acceptables. Ceci ne peut se faire sans optimisation, c'est-à-dire trouver la meilleure façon d'accéder et de traiter les données afin de répondre à la requête. Or la plupart des SGBD fournissent des langages non procéduraux (assertionnels, algébriques) qui ne fournissent pas les chemins d'accès aux données, ainsi l'optimisation devient l'affaire du système et échappe à l'utilisateur qui formule sa requête. Pour cela, le SGBD dispose d'un module d'évaluation de questions qui permet de choisir la meilleure stratégie possible d'exécution d'une expression relationnelle.

L'évaluation de questions peut être divisée en trois grandes phases :

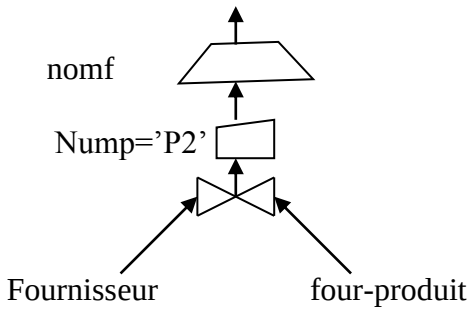
- **Analyse de la question** et élaboration d'un arbre de requête indépendamment des valeurs et des chemins d'accès aux données.
- **Ordonnancement des opérations** élémentaires (opérations algébriques) en utilisant des règles de transformation d'expression, afin de construire des plans de requête.
- **Estimation des coûts d'exécution** des plans de requête obtenus dans l'étape précédente afin de pouvoir choisir le moins coûteux.

Exemple fixant la problématique :

Nous travaillerons sur la base de données des fournisseurs avec les hypothèses suivantes :
Cardinalité fournisseurs = 100, cardinalité four-produit = 10000
Soit la requête suivante : « donner les noms des fournisseurs de la pièce P2 »
La requête SQL est la suivante :

```
SELECT nomf
FROM fournisseur, four-produit
Where nump = 'P2' and fournisseur.numf= four-produit.numf
```

Un arbre algébrique non optimisé de la requête est le suivant :



Evaluation de l'arbre non optimisé :

- 1ière étape : jointure des relations fournisseur et four-produit :
  - lecture des 10000 tuples de four-produit, puis lecture de chacun des 100 fournisseurs 10000 fois, construction du résultat intermédiaire de jointure (10000 tuples joints). Il n'est pas sûre que celle-ci puisse être gardée en mémoire.
- 2ième étape : restriction du résultat de l'étape précédente

- les seuls tuples vérifiant nump='P2' seront gardés. Nous supposons qu'ils sont 50 et peuvent être gardés en mémoire.
- 3ième étape : projection sur le nomf
  - parcours des 50 tuples du résultat précédent afin de ne garder que nomf.

Nous constatons la perte de temps en effectuant la jointure en premier du fait des lectures répétées.
La stratégie suivante est beaucoup plus efficace :

- 1ère étape : effectuer la restriction de four-produit à nump= 'P2'
  - lecture des 10000 tuples de four-produit pour ne garder en mémoire qu'une table de 50 tuples
- 2ième étape : effectuer la jointure du résultat précédent avec fournisseur
  - cette étape entraîne l 'extraction des 100 fournisseurs . Le résultat contient 50 tuples
- 3ième étape : effectuer la projection

Ainsi la commutation de la jointure et la restriction réduit considérablement le nombre de transferts de la mémoire secondaire vers la mémoire centrale, donc implique un gain de temps et donc une réduction du coût. Nous verrons plus formellement cette stratégie dans le paragraphe suivant.

2. L'ANALYSE DE QUESTION

Objectif : étude syntaxique de la question, c'est-à-dire vérification de sa correction et parfois une analyse sémantique est effectuée. Lors de cette étape, il y a vérification de la présence des attributs référencés avec les catalogues.

Résultat : génération d'un arbre de requêtes. Un arbre de requêtes est construit en utilisant une syntaxe abstraite associée aux opérateurs de l'algèbre relationnelle ou au calcul relationnel. Pour des raisons de commodité et afin de faciliter la compréhension , nous travaillerons sur une syntaxe proche de la notation utilisée dans l'algèbre relationnel. L'arbre de requêtes sera alors un arbre algébrique.

Un arbre algébrique est un arbre dont :

- les nœuds terminaux sont des relations,
- les nœuds intermédiaires sont des opérateurs de l'algèbre relationnelle ,
- le nœud racine est le résultat de la question,
- les arcs sont les flux de données entre les opérateurs.

L'analyse sémantique consiste à vérifier si la question est bien formulée comme par exemple s'il y a oubli d'un opérateur ou s'il y a des contradictions (sélection avec deux critères >10 et <10 par exemple).

3. ORDONNANCEMENT DES OPERATIONS

L'ordonnancement des opérations relationnelles est complexe. Nous ne verrons dans cette partie qu'un ordonnancement bas é sur une restructuration algébrique. L'optimisation dépend en grande partie de l'ordre d'exécution des opérateurs apparaissant dans l'arbre algébrique utilisé.

Stratégie générale d'une optimisation

- Traduire les sous-arbres booléens en une opération ayant un prédicat composé.

- Effectuer les opérations unaires de restriction et projection le plus tôt possible car elles réduisent respectivement le degré et la cardinalité d'une relation.
- Pré-traiter les relations avant d'effectuer la jointure soit en créant des index soit en effectuant le tri.
- Rechercher les sous expressions communes à plusieurs sous-arbres pour les pré-évaluer.
- Traiter en une seule fois les opérations de restriction et projection sur une même relation.

3.1. REGLES DE TRANSFORMATION DES ARBRES

3.1.1. Règles de regroupement

Règle 1 : regroupement des projections

$$\pi\ (\pi\ (R/\ A1, A2, \ldots, An))\ /\ B1, B2, \ldots, Bp) = \pi\ (R\ /\ B1, B2, \ldots, Bp)$$

avec $\{Bj\} \subset \{Ai\}$

Règle 2 : regroupement des restrictions

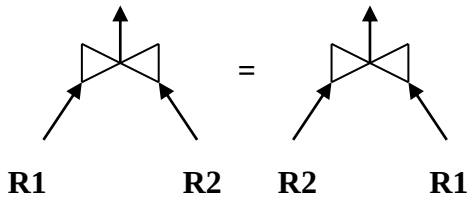
$$\sigma(\sigma\ (\ R\ /\ Xi\ \theta\ a)/\ Xj\ \theta\ b\ ) = \sigma\ (\ R\ /\ Xi\ \theta\ a \wedge Xj\ \theta\ b)$$

3.1.2. Règles de commutativité

Règle 3 : Commutativité des jointures et produit cartésien

R1 ⋈ R2 = R2 ⋈ R1

Ou ⊗ **ou** ⊗



Les opérateurs d'union et d'intersection sont aussi commutatifs, la différence ne l'est pas.

3.1.3. Règles de commutation

La distributivité d'un opérateur Θ sur un opérateur θ s'exprime comme suit :
Θ (R1θ R2) = (Θ R1) θ (Θ R2)

Règle 4 : Commutation des restrictions et projections

1er cas : l'argument de restriction fait partie des attributs de projection

$$\pi/(X1,\ldots, ,Xi)\ (\sigma\ (R/\ (Xp\ \theta\ a\ )) = \sigma/\ (\ Xp\ \theta\ a\ )\ (\ \pi(R\ /\ (X1,\ldots, Xp\ldots,Xi))$$

2ième cas : Sinon

π /(X1,…, ,Xi) (σ (R/ (Xp θ a)) = π)/ X1,…,Xi (σ/ Xp θ a (π (R/(X1,…,Xi, Xp)))

Règle 5 : Commutation des restrictions avec l'union

σ (**Union** (R1, R2) / Xp θ a) = **Union** (σ(R1/ Xp θ a), σ(R2/ Xp θ a))

Règle 6 : Commutation des restrictions avec la différence

σ (**Minus** (R1, R2) / Xp θ a) = **Minus** (σ(R1/ Xp θ a), σ(R2/ Xp θ a))

Règle 7 : Commutation des restrictions avec le produit cartésien

1er cas : la restriction porte sur l'argument d'une seule relation : les deux relations n'ont pas d'attributs communs : R1 (…., Xi, …), R2 (…., Yj, …)

σ (R1 $\otimes$ R2) / (Xi θ a)) = σ (R1/ (Xi θ a)) $\otimes$ R2

2ième cas : la restriction porte sur deux arguments respectifs de R1 et R2 , les deux relations n'ont pas d'attributs communs : R1 (…., Xi, …), R2 (…., Yj, …)

σ (R1 $\otimes$ R2) / (Xi θ a $\wedge$ Yj θ b)) = σ (R1/ (Xi θ a)) $\otimes$ σ (R2/ (Yj θ b))

3ième cas : la restriction porte sur deux arguments respectifs de R1 et R2 , les deux relations ont un attribut commun Xp : R1 (…., Xi, Xp, …), R2 (…., Xp, …)

σ (R1 $\otimes$ R2) / (Xi θ a $\wedge$ Xp θ b)) = σ ((σ (R1/ (Xi θ a)) $\otimes$R2)/ (Xp θ b))

Règle 8 : Commutation des projections et produit cartésien
Soient R1 (…., Xi, …) et R2 (…., Yj, …)

$$\pi((R1 \otimes R2) / X1, \ldots, Xp, Y1, \ldots Yp) = \pi(R1 / X1, \ldots, Xp) \otimes \pi(R2 / Y1,\ldots,Yp)$$

Règle 9 : commutation des projections avec l'union

$$\pi((R1 \textbf{ Union } R2) / X1, \ldots, Xp) = \pi(R1 / X1, \ldots, Xp) \textbf{ Union } \pi(R2 / X1,\ldots,Xp)$$

Règle 10 : Cas de la jointure naturelle
Une jointure naturelle de R1 (X1…., Xi, …,Xn), R2 (Y1,…., Xi, …,Ym) est une restriction d'un produit cartésien à Xi = Xi suivie de la projection sur R1R2
Commutation avec la projection :

$$\pi((R1 \bowtie R2) / X1….XpY1…Yq) = \pi(\pi(R1 / X1, \ldots, Xp,Xi) \bowtie \pi(R2 / Y1,\ldots,Yp,Xi) / X1..XpY1..Yq)$$

Remarque : la dernière projection n'est nécessaire que si Xi n'appartient pas à X1,…Xp ou Y1,..,Yq
Commutation avec la restriction : R1 (…,Xi ,…) et R2 (…,Xi,..Yj,…)

$$\sigma((R1 \bowtie R2) / (Yj\ \theta\ b)) = R1 \bowtie (\sigma(R2 / (Yj\ \theta\ b))$$

A toutes ces règles peuvent être rajoutées des **règles d'associativité.**
L'associativité est définie comme suit :
$A\Theta (B\Theta C) = (A\Theta B) \Theta C$
L'union, l'intersection, et la jointure sont toutes associatives. La différence ne l'est pas.

3.1.4. Algorithme simple de génération d'un plan de requête

Un algorithme simple d'utilisation des règles précédemment décrites afin de générer des plans de requêtes est le suivant : [Miranda p 246]

- Appliquer les règles 1 et 2 pour regrouper les restrictions et projections
- Faire descendre les restrictions le plus bas possible à l'aide de la règle R4 (pour passer une projection), de R5 et R6 (pour passer une union ou une différence) de R7 (pour passer un produit ou une jointure) et combiner éventuellement des restrictions avec R2
- Faire descendre les projections à travers les projections en utilisant la règle R1 , à travers les restrictions (R4), à travers les produits (R8) , à travers les jointures (R10)
- Supprimer les projections redondantes :
  - Au niveau des feuilles de l'arbre de requêtes si tous les attributs sont cités
  - Au niveau des nœuds par l'utilisation de la règle de regroupement R1

Plusieurs plans de requêtes peuvent être généré par transformations d'expressions. L'optimiseur devra choisir alors le moins coûteux. C'est l'étape d'estimation des coûts d'exécution qui permettra de choisir.

4. ESTIMATION DES COUTS D'EXECUTION

Une solution simple pour choisir le meilleur plan d'exécution consiste à les générer tous et à estimer le coût d'exécution de chacun et choisir le moindre coût. Malheureusement cette démarche n'est pas très réalisable car :

- le nombre de plans peut être très grand
- la taille des résultats intermédiaires pris en compte est un surcoût.

Une solution simple consiste à utiliser des connaissances statistiques stockées, et des connaissances sur les coûts d'implantation des opérateurs, afin de choisir la meilleurs stratégie d'exécution.

4.1. UTILISATION DE DONNEES STATISTIQUES

L'estimation du coût d'exécution utilisent les statistiques de la base de données mémorisées dans les catalogues. Nous rappelons que certaines informations comme la cardinalité des relations existent déjà dans les catalogues déjà vus. Certaines informations complémentaires sont disponibles comme le nombre de pages occupées par une table, le nombre distinct de valeurs dans une colonne, le nombre de niveaux d'index , le nombre de pages de chaque niveau …

4.2. UTILISATION DE PROCEDURES D'IMPLANTATION DES OPERATEURS

Le programme d'optimisation va avoir à sa disposition un ensemble de procédures d'implantation prédéfinies des opérateurs.

Par exemple : la restriction pourra être implémentée différemment suivant que la formule de qualification fait référence à une clé, ou à un attribut non clé mais indexé, ou tout simplement à un attribut non clé non indexé.

La jointure de deux relations R1 et R2 peut être implémentée brutalement à l'aide de deux boucles imbriquées (une boucle externe complète sur les occurrences de R1 et une boucle interne complète sur les occurrences de R2),

ou s'il existe un index sur l'attribut de jointure, il ne sera pas nécessaire d'effectuer la deuxième boucle mais effectuer un accès direct au tuple (recherche indexée)

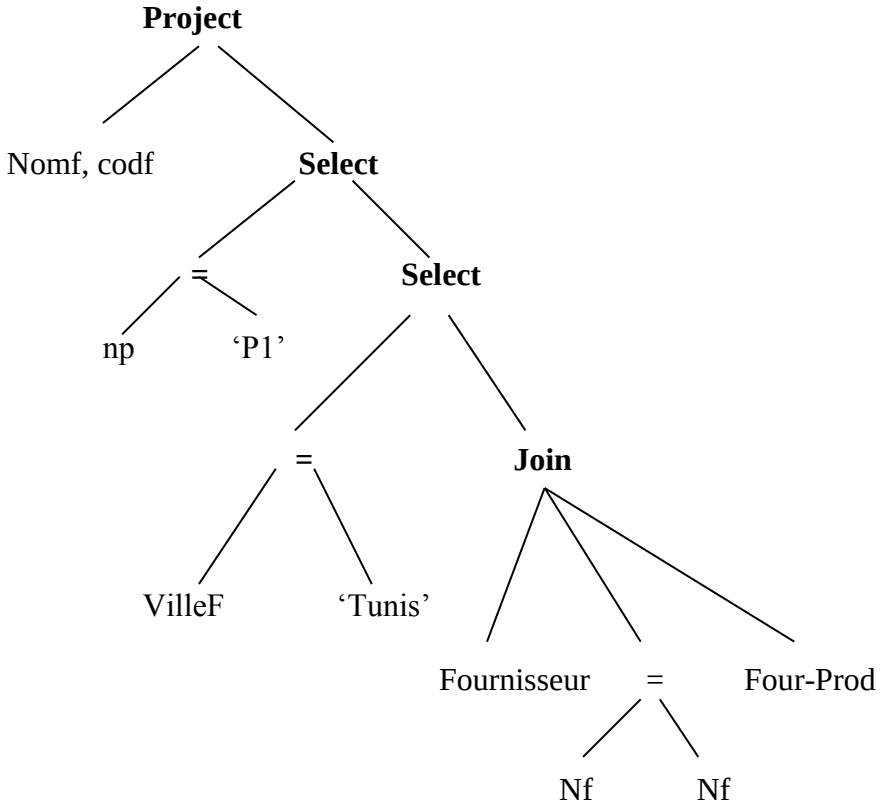
ou encore il sera possible de trier les deux relations suivant l'attribut de jointure, ainsi le parcours des deux relations pourra être synchronisé et ne fournira qu'un seul parcours de données sans redondance (implémentation par fusion).
Chaque procédure aura une formule de coût paramétrée qui pourra ainsi guider l'optimiseur dans son choix.

Exemple : soit la base de données suivante :

Fournisseur (nf, nomf, codf, villef)
Produit (np, désignation, villep)
Four-prod (nf, np, qte)

Et soit la requête :

Select nomf, codf
From Fournisseur, Four-Prod
Where Fournisseur.nf = Four-Prod.nf and VilleF = 'Tunis' and np = 'P1'

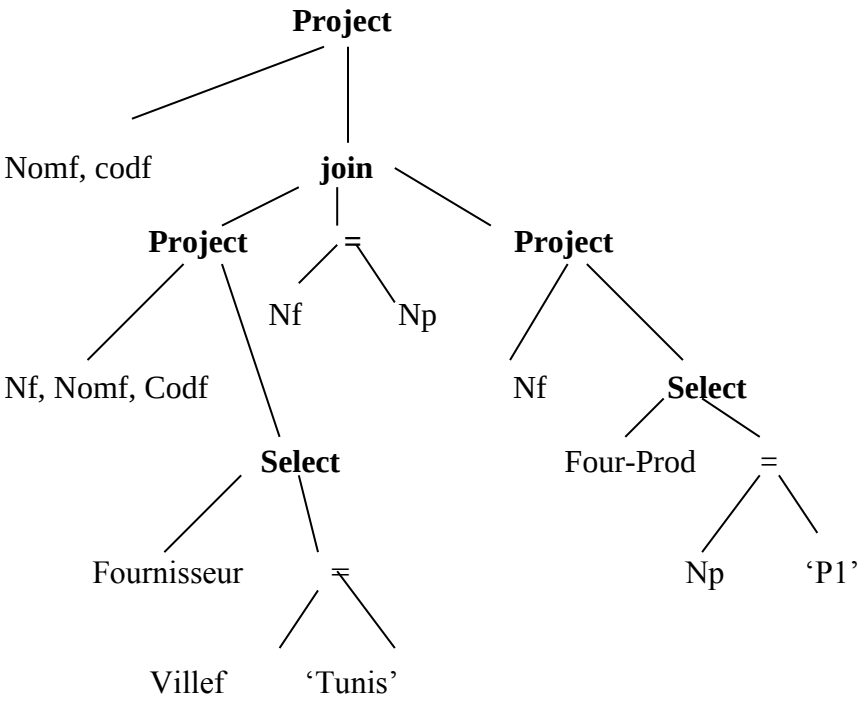


Une séquence SQL peut donner lieu à différentes interprétations qui peuvent conduire à différents temps d'exploitation. L'étape d'optimisation a lieu qui consiste en :

- grâce aux différents catalogues qui décrivent la base (la méta base), le système peut faire la correspondance entre les noms symboliques (noms de relations, d'attributs) et les objets effectivement gérés par le système (analyse sémantique).
- Le système vérifie que l'usager qui a émis la requête va manipuler les objets qu'il est effectivement autorisé à manipuler (gestion des droits d'accès et de la confidentialité).
- Un ordre peut faire référence à une relation de base ou à une vue. La vue est définie par référence à des relations de base. Il convient alors de remplacer la référence à la vue par une référence aux relations de base (phase de substitution).

- Déterminer les chemins d'accès disponibles et la meilleure stratégie d'accès aux données. Pour cela, le système dispose dans ces catalogues des index définis sur chaque relation mais aussi d'informations statistiques sur les relations à manipuler.

Résultat de l'étape : arbre optimisé à partir duquel il y a exécution. L'interpréteur SQL fournit le résultat de la requête en exécutant les opérateurs contenus dans l'arborescence à l'aide de procédures associées à ces différents opérateurs.

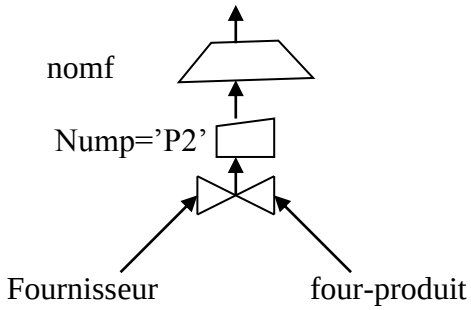


Exemple fixant la problématique :

Nous travaillerons sur la base de données des fournisseurs avec les hypothèses suivantes :
Cardinalité fournisseurs = 100, cardinalité four-produit = 10000
Soit la requête suivante : « donner les noms des fournisseurs de la pièce P2 »
La requête SQL est la suivante :

```
SELECT nomf
FROM fournisseur, four-produit
Where nump = 'P2' and fournisseur.numf= four-produit.numf
```

Un arbre algébrique non optimisé de la requête est le suivant :



Evaluation de l'arbre non optimisé :

- 1ière étape : jointure des relations fournisseur et four-produit :
  - lecture des 10000 tuples de four-produit, puis lecture de chacun des 100 fournisseurs 10000 fois, construction du résultat intermédiaire de jointure (10000 tuples joints). Il n'est pas sûre que celle-ci puisse être gardée en mémoire.
- 2ième étape : restriction du résultat de l'étape précédente
  - les seuls tuples vérifiant nump='P2' seront gardés. Nous supposons qu'ils sont 50 et peuvent être gardés en mémoire.
- 3ième étape : projection sur le nomf
  - parcours des 50 tuples du résultat précédent afin de ne garder que nomf.

Nous constatons la perte de temps en effectuant la jointure en premier du fait des lectures répétées.
La stratégie suivante est beaucoup plus efficace :

- 1ère étape : effectuer la restriction de four-produit à nump= 'P2'
  - lecture des 10000 tuples de four-produit pour ne garder en mémoire qu'une table de 50 tuples
- 2ième étape : effectuer la jointure du résultat précédent avec fournisseur
  - cette étape entraîne l 'extraction des 100 fournisseurs . Le résultat contient 50 tuples
- 3ième étape : effectuer la projection

Ainsi la commutation de la jointure et la restriction réduit considérablement le nombre de transferts de la mémoire secondaire vers la mémoire centrale, donc implique un gain de temps et donc une réduction du coût. Nous verrons plus formellement cette stratégie dans le paragraphe suivant.