Evaluation et optimisation de requêtes

Introduction

La plupart des SGBD relationnels offrent aujourd'hui des langages de manipulation basés sur SQL, non procéduraux et utilisant des opérateurs ensemblistes. Avec de tels langages, l'utilisateur définit les données qu'il veut utiliser sans fournir les algorithmes d'accès aux données. Le but des algorithmes d'optimisation de requêtes est justement de déterminer les algorithmes d'accès. Classiquement, un optimiseur transforme une requête exprimée dans un langage source (SQL) en un plan d'exécution composé d'une séquence d'opérations de bas niveaux réalisant efficacement l'accès aux données.

1- Traitement d'une requête :

Le traitement d'une requête s'effectue en trois étapes :

a) Analyse de la requête : qui correspond à l'analyse lexicale et synthaxique (ex : vérifier l'existence des tables, des attributs, ….) et quelques vérification sémantiques (ex : pas de critères contradictoires, …). Résultat de cette étape : **plan d'execution logique** qui représente la requête sous la forme d'un arbre algébrique (arbre ayant pour nœuds les opérations de l'algébre relationnelle, pour feuilles les relations et dont les arcs représentent les flux de données). Remarque : plusieurs arbres différents permettent de modéliser la même requête.

b) Optimisation : le plan logique est transformé en lui appliquant :

- Un réarrangement de l'ordre des opérations supposé optimal. Cette étape est indépendante des données.
- Un remplacement des opérateurs algébriques par des algorithmes (stratégies). Ce choix est effectué à partir de l'évaluation des couts des différentes stratégies possibles en fonction des caractéristiques des fichiers sur lesquels sont implantés les relations : remplacement des opérateurs par des algorithmes choisis de manière à optimiser l'exécution (estimation des couts), on obtient alors un **plan d'exécution physique**.

c) Exécution : le plan d'exécution physique est transformé en un programme qui une fois exécuté fournit le résultat de la requête.

2- Optimisation d'expressions algébriques (optimisation logique)

Cette étape est également appelée étape de réecriture ou restructuration algébrique. Il s'agit dans cette étape de traduire la requête utilisateur en une forme interne (canonique) basée sur l'algèbre relationnelle. L'optimisation consiste ensuite à déterminer l'expression algébrique équivalente à la requête initiale dont l'exécution sera plus rapide. On utilise pour cela les propriétés des opérateurs algébriques qui permettent de transformer une requête sans en changer le résultat. L'objectif est de trouver un arbre optimal en appliquent un certains nombre de règles et de propriétés.

2.1- Equivalence d'expressions :

Soient E_1, E_2, E_3 des expressions relationnelles, soient F_1 et F_2 des conditions et $\otimes$ l'opérateur de jointure

a) La commutativité :

$E_1 \otimes_{F_1} E_2 \Leftrightarrow E_2 \otimes_{F_1} E_1$

$E_1 \otimes E_2 \Leftrightarrow E_2 \otimes E_1$

$E_1 \times E_2 \Leftrightarrow E_2 \times E_1$

b) Associativité :

$(E_1 \otimes E_2) \otimes E_3 \Leftrightarrow E_1 \otimes (E_2 \otimes E_3)$

$(E_1 \times E_2) \times E_3 \Leftrightarrow E_1 \times (E_2 \times E_3)$

c) Règles pour les restrictions :

La restriction est un opérateur crucial pour l'optimisation parce qu'il réduit la quantité de données traitées. L'une des régles les plus importantes est de pousser (tant que possible) les selections vers le fond de l'arbre algébrique (c'est-à-dire vers les feuilles de l'arbre).

- Régle de regroupement/décomposition

$\sigma_{F_1 \Lambda F_2}(E) \Leftrightarrow \sigma_{F_1}(\sigma_{F_2}(E))$

$\sigma_{F_1 V F_2}(E) \Leftrightarrow \sigma_{F_1}(E) \cup \sigma_{F_2}(E)$

$\sigma_{F_1}(\sigma_{F_2}(E)) \Leftrightarrow \sigma_{F_2}(\sigma_{F_1}(E))$ (commutativité)

- Distribution de la restriction par rapport au produit cartésien et à la jointure

La restriction doit être avancée uniquement à l'opérateur dont les attributs sont concernés par la condition. En supposant que tous les attributs de F sont des attributs de E1 :

$\sigma_F(E_1 \times E_2) \Leftrightarrow \sigma_F(E_1) \times E_2$

$\sigma_F(E_1 \otimes E_2) \Leftrightarrow \sigma_F(E_1) \otimes E_2$

$\sigma_F(E_1 \otimes_{F_1} E_2) \Leftrightarrow \sigma_F(E_1) \otimes_{F_1} E_2$

Si F est une condition qui ne combine que des attributs partagés par E_1 et E_2, alors :

$\sigma_F(E_1 \otimes E_2) \Leftrightarrow \sigma_F(E_1) \otimes \sigma_F(E_2)$

- Distribution de la restriction par rapport à l'union /intersection : il s'agit d'avancer la restriction aux deux arguments

$\sigma_F(E_1 \cup E_2) \Leftrightarrow \sigma_F(E_1) \cup \sigma_F(E_2)$

$\sigma_F(E_1 \cap E_2) \Leftrightarrow \sigma_F(E_1) \cap \sigma_F(E_2)$

- Distribution de la restriction par rapport à la différence

$\sigma_F(E_1 - E_2) \Leftrightarrow \sigma_F(E_1) - \sigma_F(E_2)$

ou

$\sigma_F(E_1 - E_2) \Leftrightarrow \sigma_F(E_1) - E_2$

d) Règles pour les projections :

Pousser les projections vers les feuilles peut aussi être très utile (mais moins que pour les restrictions).

Pousser une projection revient à introduire une nouvelle projection sans changer la projection initiale. La règle générale set la suivante :

On peut introduire une projection n'importe où dans le plan d'exécution, pourvu qu'elle élimine uniquement des attributs non utilisés par les opérateurs suivants et qui ne sont pas dans le résultat final.

$$\pi_{A1,A2,...,An}(E) \Leftrightarrow \pi_{A1,A2,...,An}(\pi_{B1,B2,...,Bm}(E))$$

A1,A2,…,An doivent être inclus dans B1,B2,…,Bm

- Inversion restriction-projection :

Si la condition F ne combine que les attributs A1, …,An, alors :

$$\pi_{A1,A2,...,An}(\sigma_F(E)) \Leftrightarrow \sigma_F(\pi_{A1,A2,...,An}(E))$$

Plus généralement, si la condition F combine les attributs B1,..,Bm qui ne sont pas parmi les A1,…,An , alors :

$$\pi_{A1,A2,...,An}(\sigma_F(E)) \Leftrightarrow \pi_{A1,A2,...,An}(\sigma_F(\pi_{A1,A2,...,An,B1,...,Bm}(E)))$$

- Distribution projection-produit cartésien

Si A1,…,An est une liste d'attributs dans laquelle B1,…,Bm sont des attributs de E1 et les attributs restants C1,…,Ck sont de E2, alors :

$$\pi_{A1,A2,...,An}(E_1 \times E_2) \Leftrightarrow \pi_{B1,...,Bm}(E_1) \times \pi_{C1,...,Ck}(E_2))$$

- Distribution projection-union

$$\pi_{A1,...,An}(E_1 \cup E_2) \Leftrightarrow \pi_{A1,...,An}(E_1) \cup \pi_{A1,...,An}(E_2))$$

2.2- Algorithme d'optimisation d'expressions algébriques :

Le principe de base est de pousser (descendre) les opérateurs unaires au maximum au niveau de l'arbre. En effet, les opérateurs unaires sont des réducteurs de la taille des relations.

L'algorithme est le suivant :

1- Séparer les restrictions comportant plusieurs prédicats à l'aide des règles R1 et R2 appliquées de la gauche vers la droite
2- Descendre les restrictions aussi bas que possible en appliquent les règles R4 … R10
3- Regrouper les restrictions successives portant sur une même relation à l'aide des règles R1 et R2 appliques cette fois de droite vers la gauche
4- Descendre les projections aussi bas que possible en appliquant les règles R11 … R14
5- Regrouper les projections successives sur la même relations et éliminer d'éventuelles projections qui auraient pu apparaitre (projection sur tous les attributs d'une table)

Remarque : L'étape 3 et l'étape 5 permettent de diminuer les accès à la relation.

3- Evaluation des coûts et optimisation physique

Dans cette phase, il s'agit de remplacer les opérateurs algébriques par des algorithmes (appelés également stratégies). Le choix de l'algorithme dépend entre autre de la taille des relations (nombre d'accès E/S) et de la prise en compte des chemins d'accès aux relations.
Donc dans cette phase, le SGBD procède au calcul du coût d'une opération algébrique en fonction de la taille des relations, de l'algorithme choisi et des chemins d'accès.
Une solution simple pour l'optimisation physique consiste à utiliser les connaissances statistiques stockées, et des connaissances sur les coûts d'implantation des opérateurs (algorithmes) afin de choisir le meilleur plan d'exécution physique.

a) Utilisation des données statistiques :

Pour optimiser la requête, l'optimiseur utilise les données statistiques de la BD mémorisées dans les catalogues telles que : la cardinalité des relations, le nombre de pages occupées par une relation, le nombre de niveaux d'index, le nombre de pages de chaque niveau, etc.

b) Utilisation des algorithmes (procédures) d'implantation des opérateurs :

L'optimiseur aura a sa disposition un ensemble d'algorithmes pour chaque opérateur algébrique, il aura à choisir l'algorithme le moins coûteux (en terme de nombre d'accès d'E/S).

1- Opérateur de sélection : Pour la sélection, elle peut être implémentée différemment suivant que la formule de qualification fait référence à une clé, ou un attribut non clé mais indexé, ou tout simplement un attribut non clé non indexé (balayage séquentiel).

1.1- Sélection sans index : le balayage séquentiel nécessite de comparer chaque tuple de la relation opérande avec le critère. Si celui-ci est vrai, les attributs utiles sont conservés en résultat, la procédure effectue donc à la fois restriction et projection. Deux cas peuvent se présenter :

- Le fichier est non trié sur l'attribut de restriction : le balayage séquentiel complet du fichier est alors nécessaire. Cout de l'opération=page(R), R : relation opérande, page(R) : Nombre de pages du fichier
- Le fichier est trié sur l'attribut de restriction : une recherche dichotomique est possible, cout de l'opération : $Log_2(page(R))$

1.2- Sélection avec index de type arbre B : un index associe les valeurs d'un attribut avec la liste des adresses de tuples ayant cette valeur. Souvent l'index est organisé en arbre B+, dans ce nas le cout de l'opération= nombre de niveaux de l'arbre B+ 1 accès au bloc contenant la valeur recherchée (cas de la restriction attribut=valeur)

2- Opérateur de jointure : Pour le jointure de R1 et R2, il est possible de distinguer différents algorithmes selon la présence d'index sur les attributs de jointure ou non. On peut citer les algorithmes suivants :

2.1- Algorithme des boucles imbriquées (une pour parcourir R1, et une boucle interne pour parcourir R2) : c'est l'algorithme le plus simple. Il consiste à lire séquentiellement la première relation R1 et à comparer chaque lu avec chaque tuple de R2. R1 est appelée relation externe et R2 relation interne.

Algorithme :

```
Pour chaque page B1 de R1 faire
    Lire (R1, B1) ;
    Pour chaque page B2 de R2 faire
        Lire (R2, B2) ;
        Pour chaque tuple tuple1 de R1 faire
            Pour chaque tuple tuple2 de R2 faire
                Si tuple1 et tuple2 sont joignable
                alors écrire (Resultat, tuple1+tuple2) ;
            fin
        fin
    fin
fin.
```

Les entrées-sorties s'effectuant par page, en notant Page(R) le nombre de pages d'une relation R, on obtient un cout en entrée-sortie de : Cout(opération)=Page(R1)+Page(R1)*Page(R2)
Si la mémoire cache permet de mémoriser (B+1) pages, il est possible de garder B pages de R1 et de lire R2 seulement Page(R1)/B fois, la formule devient alors:
Cout(opération)=Page(R1)+Page(R1)*Page(R2)/B
Remarque: il est plus intéressant de choisir comme relation R1 celle qui a le plus petit nombre de blocs.

2.2-- Algorithme de tri-fusion qui consiste à trier les deux relations suivant l'attribut de jointure, ensuite la fusion des tuples ayant même valeur est effectuée (parcours synchronisé des deux relations) (cas de l'equi jointure). En supposant que le cout d'un tri (tri binaire) de N pages est $2N \log_2(N)$, le cout de la jointure sera: 2*Page(R1) Log(Page(R1))+ 2*Page(R2) Log(Page(R2))+Page(R1)+Page(R2), en considérant une fusion simple c'est à dire un seul parcours des deux relations.

2.3- jointure avec index (cas ou l'une des relations est indexée sur l'attribut de jointure par exemple R1) dans ce cas il suffit de balayer la deuxième relation R2 et accéder au fichier index pour chaque tuple. En cas de succès, on procède à la concaténation pour composer le tuple résultat. Le cout est de l'ordre de n*Tuple(R2), n étant le nombre d'accès en moyenne pour trouver un article dans le fichier indexé, et tuple(R2) est le nombre de tuples de R2.
Lorsque les deux relations sont indexée sur les attributs de jointure il suffit de fusionner les deux index, les couples des adresses des tuples concaténés sont alors obtenus.

Pour chaque algorithme, il y a une formule de coût paramétrée qui pourra ainsi guider l'optimiseur dans son choix.

4- Conclusion :

L'intérêt de l'optimisation logique ou réarrangement des opérateurs :

- Le coût des opérateurs varie en fonction du volume des données traitées, c'est-à-dire plus le nombre de tuples des relations traitées est petit, plus les coûts CPU et d'E /S sont minimisées.
- Certains opérateurs diminuent le volume de données telles que la restriction et la projection.

L'intérêt de l'optimisation physique : minimiser les coûts lors de l'implémentation des opérateurs algébriques en fonction des caractéristiques des fichiers supportant les relations concernées par les opérateurs.