UNIVERSITÉ TOULOUSE III PAUL SABATIER

Rédigé avec LaTeX
Version du 20 novembre 2014

Système de Base de données Relationnelles

Optimisation de requêtes

M1 Informatique – Développement Logiciel
Semestre 7

Cours donné par Abdelkader Hameurlain
Rédigé par Antoine de Roquemaurel

2014

Avant-propos

L'objectif de l'optimisation est de minimiser le temps de recherche. Pour cela, on définit une stratégie ou méthode d'optimisation selon 3 critères :
- Plan d'exécution
- Espace de recherche
- Modèle de coûts

De ces 3 critères découlent l'obtention du Plan d'exécution le plus proche de l'optimal. (cf. schéma)

> R Le plan d'exécution n'est jamais le plus optimal possible car il engendre d'énorme coûts pour des différences moindres.

Table des matières

1 Les opérateurs physiques

1.1 Système de gestion de fichier

Definition 1.1 **Fichier** collection d'article (de fichiers) contenant de l'information (information de gestion).

Definition 1.2 SGF [a] programme permettant à l'utilisateur de :
- Définir la structure d'un fichier
- Créer un fichier
- Interroger un fichier
- Modifier un fichier

a. Système de Gestion de Fichier

1.2 Algorithme de jointure

1.2.1 Jointure par produits cartésiens

Principe : Comparaison de tous les tuples de la relation R avec tous les tuples de la relation S.

1.2.1.1 Jointure par produit cartésien sans index

$$Nombredepageslues = |R| \times |S| + |R|$$

R Pour des raisons de performance, on privilégiera la plus petite relation comme relation externe.
Relation externe = R

Inconvénients

- Nombre important d'entrées/sorties disque
- Pour chaque tuple de R la totalité de S est balayé

1.2.1.2 Jointure du produit cartésien par ind

- La création d'un index permet d'éviter que, pour chaque tuple R, la totalité de S soit balayé.
- L'utilisation d'un index se fait sur l'attribut de jointure de la relation interne (S) sous la forme (attribut, pointeur).

On obtient alors :

Index plein

$$nb_{pages} = \frac{nbTuples \times (taille(attribut) + taille(pointeur))}{taille(page)}$$

Index creux

$$nb_{pages} = \frac{nbPagesPrecedents \times (taille(attribut) + taille(pointeur))}{taille(page)}$$

Tant que l'on obtient pas un nombre de page (nbPages) égal à 1 on continue.

R On arrondi ici toujours le nombre le nombre de page à la valeur entière supérieure (on ne lit pas une moitié de page).

$$tempsLecture = (nb_{index} + 1) \times tempsAccesDisque$$

ou le nombre d'index

$$nb_{index} = 1 \times index_{plein} + n \times index_{creux}$$

R L'index réduit considérablement le temps d'exécution. Cela peut encore être amélioré en utilisant un **cluster**

1.2.1.3 Jointure par tri-fusion

Principe

- Trie les deux relations (la relation externe R et la relation interne S) sur leur attributs de jointure
- Parcourt les deux relations pour déterminer les tuples satisfaisant la condition de jointure.

1.2.1.4 Jointure par hachage

Permet de réduire le nombre de comparaison par rapport au produit cartésien.

Principe

- construit « build » une table de hachage avec la plus petite des relations (ici la relation R) La construction se fait avec une fonction de hachage appliquée sur l'attribut de jointure. Les tuples sont ainsi répartis dans des paquets « bucket » composés d'une ou plusieurs page. La fonction de hachage associe à une valeur un entier représentant un numéro de paquet dans la table. Ainsi, tous les tuples ayant même valeur après application de la fonction de hashage se retrouve dans le paquet en question.
- balaie la relation S de manière séquentielle (page par page) en sondant « build » la table de hachage afin de trouver les tuples pertinents pour calculer la jointure.

2 Optimisation logique

L'optimisation logique a pou but de retarder le plus possible les opérateurs qui augmentent l'information (comme la jointure et l'union).

Optimisation globale puis opérateur par opérateur

2.1 Principe

Préférer l'utilisation de projection/sélection AVANT une jointure/union lorsque cela est possible. Cela permet de traiter un volume de données moins important.

2.2 Calcul du volume de données

```
TRAIN(No_T, No_W)
WAGON(No_W, Type_W, Poids_W, Capacite, Etat, Gare_affect
```

Nom de la relation	Taille	Longueur d'un tuple en caractères
TRAIN	60 000	10
WAGON	200 000	30

Constitutants	Nombre de valeurs possibles	Longueur en caractères
No_T	2 000	4
No_W	200 000	6
Type_W	200	2

```
-- Liste des types de wagons
-- du train de numéro 4002
SELECT w.Type_W
FROM WAGON w, TRAIN t
WHERE t.No_W = w.No_W
AND No_T = 4002;
```

```

Volume = 60 000 * 10 + (60 000/2000) * 10 + (60 000/2000) * 6 + 200 000 * 30 + ↩
    (60 000/2000) * (30+6) + (60 000/2000) * 2

Étape 1 = taille(TRAIN) * longueur(TRAIN)

Étape 2 = moyenne * longueur(TRAIN)
= (taille(TRAIN) / nbValeursPossibles(No_T)) * longueur(TRAIN)
```

```
8
  Étape 3 =  moyenne * longueur(No_W)
10 = (taille(TRAIN) / nbValeursPossibles(No_T)) * longueur(No_W)

12 Étape 4 = nbValeursPossibles(No_W) * longueur(No_W)

14 Etape 5 = moyenne * (longueur(WAGON) + longueur(No_W)
  = (taille(TRAIN) / nbValeursPossibles(No_T)) * (longueur(WAGON) * longueur(No_W))
16
  Étape 6 = moyenne * longueur(Type_W)
18 = (taille(TRAIN) / nbValeursPossibles(No_T)) * longueur(Type_W)
```

3 Optimisation physique

3.1 Stratégies énumératives

Consiste à énumérer l'ensemble des solutions alternatives à des sous plans d'exécution déjà optimisés afin de choisir la plus performante possible.

Avantage Optimalité globale

Inconvénient Coût élevé

3.2 Stratégies aléatoires

Utilisation Lorsqu'on a un grand nombre de relations

Fonctionnement
- Obtention d'un plan d'exécution initial via une stratégie énumérative (exemple par une recherche en profondeur d'abord)
- Application de transformations générés aléatoirement au plan obtenu précédemment dans le but d'obtenir éventuellement un optimum local.

Avantages Réduction significative de la taille du problème, coût moins élevé

Inconvénient Moins optimale

4 Transactions

L'objectif des transactions est de maintenir la cohérence d'une base de données en cas d'accès concurrents et de pannes/défaillances.

4.1 Contraintes d'intégrités

L'état d'une base de données est dit cohérent s'il satisfait à toutes les contraintes d'intégrités définies.

- Contraintes statiques : Contraintes statiques : prédicat sur l'état courant de la base de données Contrainte du type primary_key, foreign_key, check

  > Clé primaire unique, salaire >= SMIC

- Contraintes dynamiques : concerne le passage d'un état à un autre Contrainte de type trigger

  > New salaire >= Old salaire

La vérification d'une contrainte d'intégrité est liée au concept de transaction. Cela peut se faire de façon immédiate ou différée.

4.2 Concept de transaction

Definition 4.1 Transaction

- Séquence d'actions qui réalise une unité logique de traitement
- Séquence d'opérations (actions) sur un ensemble d'objets

Definition 4.2 Action commande indivisible exécutée pour le compte d'une transation par le système (lire/écrire dans la base)

Propriété des transactions :

Atomicité Mise à jour de tout ou annulation de toutes les modifications

Cohérence

Isolation Visibilité de la transaction aux autres transaction qu'une fois celle-ci validée

Durabilité Les modifications validées doivent être conservées en cas de panne

Mots clés : Commit et Rollback

4.3 Synchronisation des accès concurrents

Dans un SGBD multi-utilisateurs, il se produit un conflit lorsque 2 transactions T1 et T2 s'intéressent à un même objet. Il faut donc appliquer un ordre d'exécution.

Seule l'isolation est requise pour les accès concurrents

4.4 Transactions à 2 phases

Pour chaque transaction, on doit procéder à :
- Phase d'expansion : phase d'acquisition des verrous :
  - verrous partageables (share) en lecture : v partagé(x)
  - verrous d'accès exclusif : v exclusif(x)
- Phase de réduction : phase de libération des verrous (unlock) : libérer(x)

4.5 Transaction bien formée

Une transaction T est dite bien formée lorsqu'elle obéit aux règles suivantes :
- Avant de LIRE un objet x : v partagé(x). T doit avoir au moins un verrou d'accès partagée sur x
- Avant d'ÉCRIRE sur un objet x : v exclusif(x). T doit avoir UN verrou d'accès exclusif sur x
- Aucun objet ne reste verrouillé par T après la fin de T