Evaluation et optimisation de requêtes

Patricia Serrano Alvarado
à partir de tranparents de Sylvie Cazalens et Philippe Rigaux

Evaluer, optimiser ?

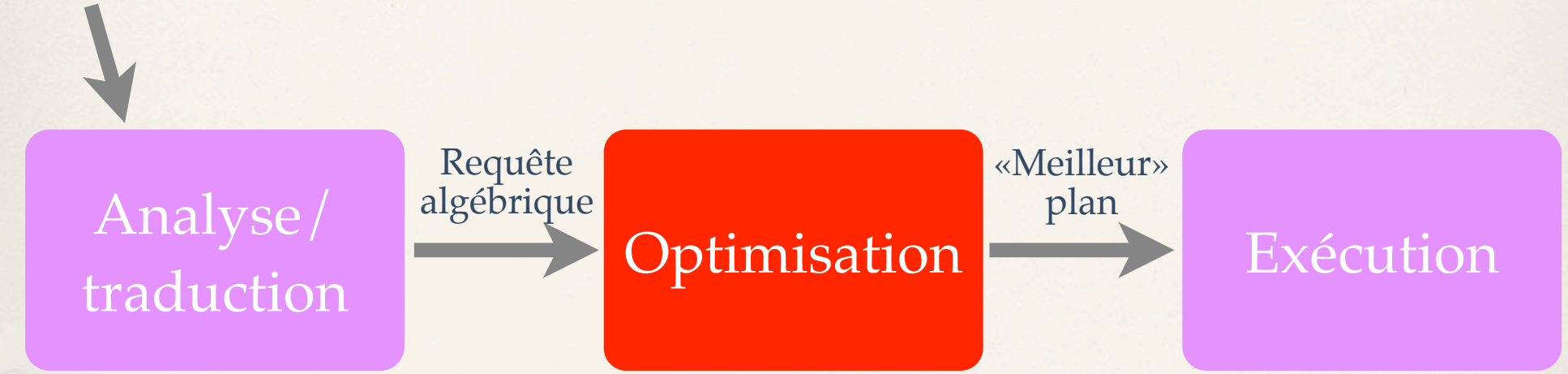
- Pour exécuter une requête SQL le SGBD doit comprendre, analyser et optimiser l'exécution
  - Passage du SQL à un ensemble d'opérations
  - Avec des «outils» on organise au mieux les opérations

Etapes du traitement d'une requête

- Toute requête SQL est traitée en trois étapes :
  - 1. Analyse et traduction de la requête. On vérifie qu'elle est correcte, et on l'exprime sous forme d'opérations.
  - 2. Optimisation : comment agencer au mieux les opérations, et quels algorithmes utiliser. On obtient un plan d'exécution.
  - 3. Exécution de la requête : le plan d'exécution est compilé et exécuté.

Principales étapes

Requête SQL
(mode déclaratif)



On obtient des opérations

On obtient un plan d'exécution

On obtient l'ensemble de tuples

Calcul d'une requête

- Requête SQL (forme déclarative)

```
SELECT nomr
FROM robots NATURAL JOIN resultats
WHERE nomc = 'WRC' AND noml = 'LAAS';
```

- Pour trouver le résultat, il faut consulter les relations stockées en mémoire secondaire. Ceci est coûteux (en temps).

- Comment calculer le résultat correctement et le plus efficacement possible ?

Exemple 1 de plan

- R1 <- Lire *séquentiellement* la table Robots et sélectionner les tuples où le nom de labo est «LAAS»
- R2 <- Lire *séquentiellement* la table Resultats et sélectionner les tuples où le nom de la compétition est «WRC»
- R3 <- Joindre R1 avec R2 sur nomr
- R4 <- Projeter R3 sur nomr

Exemple 2 de plan

- R1 <- Joindre la table Robots avec la table Résultats sur nomr
- R2 <- Lire *séquentiellement* R1 et sélectionner les tuples où le nom de labo est «LAAS»
- R3 <- Lire *séquentiellement* R2 et sélectionner les tuples où le nom de la compétition est «WRC»
- R4 <- Projeter R3 sur nomr
- Ces plans sont réprésentés en algèbre relationnelle ou avec des arbres algébriques

Support du traitement d'une requête

- Le traitement s'appuie sur les éléments suivants :
  - 1. Le schéma de la base, description des tables et chemins d'accès (dans le catalogue)
  - 2. Des statistiques : taille des tables, des index, distribution des valeurs
  - 3. Des algorithmes : il peuvent différer selon les systèmes
- Important : on suppose que le temps d'accès à ces informations est négligeable par rapport à l'exécution de la requête.

Qu'est-ce qu'optimiser ?

- Pour une même requête de nombreux arbres algébriques possibles et
- Pour chaque nœud, plusieurs algorithmes possibles
  - autant de plans différents
- L'espace des plans possibles peut être très grand
- Optimiser -> rechercher le «meilleur» plan dans l'espace des plans possibles

Optimal par rapport à quoi ?

- Temps d'exécution : temps total de traitement de la requête (throughput)
- Temps de réponse : temps d'obtention des premières réponses (response time)
- Ces deux critères dépendent
  - du nombre de transferts réalisés avec la mémoire secondaire (donc des algos)
  - de la taille des relations (nb tuples et attributs).

Estimer le coût d'un plan

- Optimiser : trouver le plan d'exécution le moins «coûteux», ou au moins éviter le pire !
- Fonction de coût : donne une estimation du coût total réel du plan considéré (coût E/S + coût CPU, ce dernier étant généralement négligeable). Pour cela on utilise des statistiques et des estimations.

Difficulté de l'optimisation

- Trouver une bonne fonction de coût
- Trouver une solution dans un espace de recherche très grand : il ne faut pas que le temps d'exécution de la requête en soit alourdit.

  Recherche non exhaustive d'une bonne solution (qui semble s'approcher de la meilleure) avec des heuristiques.

Estimer le coût d'un plan

- Estimer le nombre d'accès disque pendant l'évaluation de chaque nœud de l'arbre algébrique (pipeline en mémoire) ou relations temporaires (sur disque)
- On considère un nombre de pages disque
- Estimer la taille du résultat d'un nœud par rapport à ses entrées (sélectivité des opérations).

L'optimisation sur un exemple

- Considérons le schéma :
  - CINEMA(Cinéma,Adresse,Gérant)
  - SALLE(Cinéma,NoSalle,Capacité)
- avec les hypothèses :

1. Il y a 300 n-uplets dans CINEMA, occupant 30 pages.
2. Il y a 1200 n-uplets dans SALLE, occupant 120 pages.

Expression d'une requête

- On considère la requête : Adresse des cinémas ayant des salles de plus de 150 places

- En SQL :

```
SELECT Adresse
FROM  CINEMA, SALLE
WHERE capacité > 150 AND
CINEMA.cinéma = Salle.cinéma;
```

En algèbre relationnelle

- Traduit en algèbre, on a plusieurs possibilités. En voici deux :

1. $\pi_{Cinema}(\sigma_{Capacite>150}(CINEMA \bowtie SALLE))$
2. $\pi_{Cinema}(CINEMA \bowtie \sigma_{Capacite>150}(SALLE))$

- Soit une jointure suivie d'une sélection, ou l'inverse.
- NB : on peut les représenter comme des arbres algébriques

Évaluation des coûts

- On suppose qu'il n'y a que 5 % de salles de plus de 150 places.

1. Jointure: on lit 3600 pages (120x30); Sélection : on obtient 5 % de 120 pages, soit 6 pages. Nombre d'E/S: 3600 + 120 + 6 = 3726.

2. Sélection : on lit 120 pages et on obtient 6 pages. Jointure : on lit 180 pages (6x30) et on obtient 6 pages. Nombre d'E/S : 120 + 6 + 180 + 6 = 312.

- ⇒ la deuxième stratégie est de loin la meilleure !

Implémentation et coût des opérateurs

- Sélection sur R (attribut A)
  - parcours séquentiel (scan) : nombre d'accès disque en O(n)
  - accès avec un index B+ sur A : O(log(n))
- Projection de R
  - sans élimination des doublons : O(n)
  - avec élimination des doublons et tri : O(n log(n))

Implémentations de la jointure (1)

Soit T le résultat de la jointure entre R1 et R2 de condition R1.A = R2.B.

- **Algo. boucles imbriquées** :

Pour chaque tuple r1 de R1 faire
    pour chaque tuple r2 de R2 faire
        si r1.A=r2.B alors T <- T + <r1,r2> fin si
    fin pour
fin pour

- Coût : nombre de lectures/écritures O(n*m)

Implémentations de la jointure (2)

- **Algo boucle imbriquée avec index sur R2.B**

  Pour chaque tuple r1 de R1 faire
  pour chaque tuple r2 de R2 accédé avec $index_{R2.B}$(r1.A)
  T <- T + <r1,r2> fin si
  fin pour
  fin pour

- Coût si index sur R2.B : O(n*log(m))

Implémentations de la jointure (3)

- **Jointure par tri-fusion**

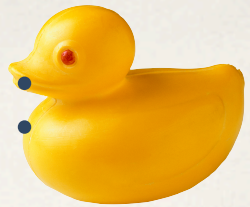
1. - trier R1 selon l'attribut de jointure A
2. - trier R2 selon l'attribut de jointure B
3. - fusionner les relations triées

- Principe de la fusion pour 2 pages :
  - indices i1 et i2 parcourant chacune des relations jusqu'à la fin de page pour l'un d'entre eux.
  - on compare les deux valeurs pointées par les indices
    - si valeurs differentes on incrémente l'indice correspondant à la valeur la plus basse
    - si valeurs égales on ajoute un tuple au résultat et on increment un indice.

Coût des opérateurs de jointure

- Boucles imbriquées : nombre de lectures/écritures O(n*m)
- Boucles imbriquées et index sur R2.B : O(n*log(m))
- Jointure par tri-fusion
  - si relations triées O(n+m)
  - si relations pas triées O(n*log(n) + m*log(m))

Estimer la taille des résultats : Statistiques



- Sur les relations
  - nombre de tuples card(R) = | R |
  - nombre d'attributs largeur ou degré(R) = δ(R)
  - fraction de tuples participant à une jointure...
- Sur les attributs
  - domaine
  - valeurs max et min, nombre de valeurs distinctes
  - distribution des valeurs
- Hypothèses usuelles : indépendance des valeurs d'attributs, distribution uniforme de ces valeurs dans leur domaine.

Taille des relations intermédiaires

- Soit R une relation
  - taille(R) = card(R)*largeur(R) où |R| * δ(R)
- SELECTION : le nombre de tuples du résultat = <u>card(R) * estimation de la ***sélectivité*** (porcentage de lignes concernées) du prédicat de sélection</u>.

$$S_{\sigma(A=valeur)} = \frac{1}{|\ \Pi_A(R)\ |}$$

$$|\ \sigma_{(A=valeur)}(R)\ | = |\ R\ | \times S_{\sigma_{(A=valeur)}}$$

$$S_{\sigma_{(A>valeur)}} = \frac{max(A) - valeur}{max(A) - min(A)}$$

Taille des relations intermédiaires

- Si la sélection est composée de plusieurs conditions :
  - sélectivité d'une **conjonction** = produit des sélectivités
  - sélectivité d'une **disjonction** : somme des sélectivités moins leur produit.
- PROJECTION : nombre de tuples inférieur ou égal à card(R).

Taille des relations intermédiaires

- Taille d'une JOINTURE R et S
  - card(R) * card(S) * estimation sélectivité de la jointure
- Cas particulier d'une condition R.A=S.B où A est clé de R et B clé étrangère de S
  - card(S)

Heuristiques pour la recherche d'un plan d'exécution

- Il y a des opérations plus ou moins
  - coûteuses
  - sélectives.
- Il vaut mieux effectuer d'abord les opérations les moins coûteuses et les plus sélectives pour diminuer la taille des entrées des opérations plus coûteuses comme la jointure.
- Heuristique : **descendre au maximum les sélections, puis les projections via transformation (équivalences algébriques).**

Equivalences algébriques

Commutativité et associativité de la jointure

$$E_1 \bowtie E_2 = E_2 \bowtie E_1,$$
$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3).$$

Cascade de projections

$$\pi_{A_1,\ldots,A_n}(\pi_{B_1,\ldots,B_m}(E)) = \pi_{A_1,\ldots,A_n}(E)$$

Cascade de sélections

$$\sigma_{F_1}(\sigma_{F_2}(E)) = \sigma_{F_1 \wedge F_2}(E)$$

Commutation sélection et projection
Si F ne porte que sur $A_1, \ldots, A_n$,

$$\pi_{A_1,\ldots,A_n}(\sigma_F(E)) = \sigma_F(\pi_{A_1,\ldots,A_n}(E))$$

Si F porte aussi sur $B_1, \ldots, B_m$,

$$\pi_{A_1,\ldots,A_n}(\sigma_F(E)) = \pi_{A_1,\ldots,A_n}(\sigma_F(\pi_{A_1,\ldots,A_n,B_1,\ldots,B_m}(E)))$$

Equivalences algébriques

Commutation sélection et $\times \cup - \bowtie$

$$\sigma_F(E_1 OP E_2) = \sigma_F(E_1) OP \sigma_F(E_2)$$

Commutation projection et $\times \cup$

$$\pi_{A_1,\ldots,A_n}(E_1 OP E_2) = \pi_{A_1,\ldots,A_n}(E_1) OP \pi_{A_1,\ldots,A_n}(E_2)$$

L'essentiel de ce qu'il faut savoir

Sur la phase d'optimisation.

- Pour une requête, le système a le choix entre plusieurs plans d'exécution.
- Ils diffèrent par l'ordre des opérations, les algorithmes, les chemins d'accès.
- Pour chaque plan on peut estimer :
  - le coût de chaque opération
    - la taille du résultat
- **Objectif** : diminuer le plus vite possible la taille des données manipulées

Laissons le choix au système !

- Bon à savoir : l'imbrication de requêtes réduit les possibilités d'optimisation.
- Exemple : cherchons tous les films avec James Stewart, parus en 1958.

```
SELECT titre
FROM  Film f, Role r, Artiste a
WHERE a.nom = 'Stewart' AND a.prenom='James' AND
f.idFilm = r.idFilm AND r.idActeur = a.idArtiste
AND f.annee = 1958
```

- Pas d'imbrication : un bloc, OK !

Seconde requête (2 blocs)

- La même, mais avec un niveau d'imbrication.

```
SELECT titre FROM  Film f, Role r
WHERE f.idFilm = r.idFilm AND f.annee = 1958 AND
r.idActeur IN (SELECT idArtiste
                FROM Artiste
                WHERE nom='Stewart' AND
                prenom='James')
```

- Une imbrication sans nécessité : moins bon !

Troisième requête (2 blocs)

- La même, mais avec EXISTS au lieu de IN.

```
SELECT titre
FROM  Film f, Role r
WHERE f.idFilm = r.idFilm ANDf.annee = 1958
AND EXISTS (SELECT 'x'
            FROM Artiste a
            WHERE nom='Stewart' AND prenom='James'
            AND r.idActeur = a.idArtiste)
```

Quatrième requête (3 blocs)

- La même, mais avec deux imbrications :

```
SELECT titre FROM Film
WHERE annee = 1958 AND idFilm IN
                (SELECT idFilm FROM Role
                WHERE idActeur IN (SELECT idArtiste
                                    FROM Artiste
                                    WHERE nom='Stewart'
                                   AND prenom='James'))
```

- Très mauvais : on force le plan d'exécution, et il est très inefficace.

Pourquoi c'est mauvais

- On parcourt tous les films parus en 1958
- Pour chaque film : on cherche les rôles du film, mais pas d'index disponible
- Ensuite, pour chaque rôle on regarde si c'est James Stewart
- Ca va coûter cher !!

Conclusion

- Il faut connaître les grandes lignes du processus d'évaluation d'une requête, sachant que pour des raisons de performances (tuning) il faudra peut-être étudier très en détail les algos.
- L'optimisation de requêtes est un problème difficile qui est toujours d'actualité car
  - les types de requêtes évoluent et il faut trouver de nouvelles techniques (OLAP, PicoDBMS)
  - les modèles de BD changent (xml, rdf...)