Bases de données (CS443)

#5, Modèle relationnel: optimisation de requêtes

Laure Gonnord

Grenoble INP/Esisar

2022-2023

Crédits

Transparents F. Duchateau, pour univ Lyon1, CC by SA.

https://perso.liris.cnrs.fr/fabien.duchateau/BDW1/

Motivation

Motivation (2)

Une requête SQL est exécutée par le SGBD, mais :

- Quel est l'impact des différentes manières d'écrire une requête pour un même résultat (e.g., NOT IN / NOT EXISTS) ?
- Comment passe t-on de la requête (définie dans un langage déclaratif) à un programme (impératif) manipulant les données ?
- Comment optimise t-on une requête afin de l'exécuter efficacement pour trouver un résultat correct ?
- Pourquoi les requêtes préparées permettent de gagner en performance (entre autre) ?

---

http://sqlpro.developpez.com/cours/optimiser/

Optimiseurs

L'optimiseur de requêtes d'un SGBD est un composant crucial :

- Il réécrit la requête sous différentes formes
- Il choisit la réécriture la plus performante pour exécuter la requête
- Des benchmarks évaluent et comparent les SGBD, notamment les performances de leur optimiseur

---

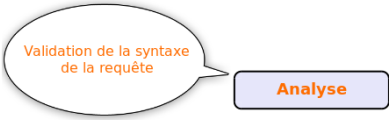
Jarke, Matthias, and Jurgen Koch. *Query optimization in database systems*. ACM Computing surveys (1984)

Graefe Goetz. *Query evaluation techniques for large databases*. ACM Computing Surveys (1993)

Ioannidis, Yannis. *Query Optimization*. The Computer Science and Engineering Handbook (1996)

Plan

Traitement d'une requête

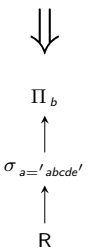


Où l'on retrouve l'algèbre relationnelle...

Une requête en algèbre relationnelle peut se représenter sous forme d'arbre algébrique

- Racine de l'arbre = résultat de la requête
- Feuilles de l'arbre = relations
- Noeuds intermédiaires de l'arbre = un opérateur algébrique (e.g., sélection, union, jointure)

$\Pi_b(\sigma_{a='abcde'}(R))$



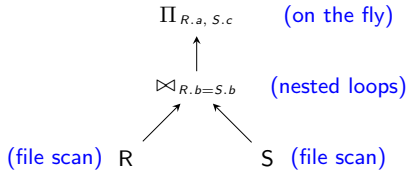
Arbre algébrique ≈ plan d'exécution d'une requête

Où l'on retrouve l'algèbre relationnelle... (2)

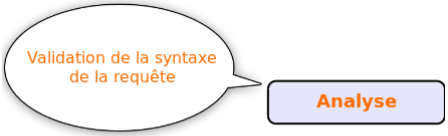
- Le plan d'exécution d'une requête est généralement optimisé selon trois opérateurs (projection, sélection et jointure)
- Les autres opérateurs (e.g., regroupement, tri) sont réalisés ensuite (ajoutés au plan optimisé des trois opérateurs)
- Un plan d'exécution indique quel algorithme est appliqué pour chaque opérateur

SELECT R.a, S.c
FROM R, S
WHERE R.b = S.b ;

$\Longrightarrow$



Étape d'analyse

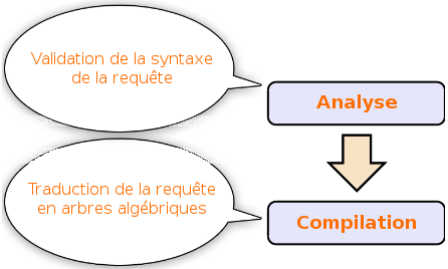


Étape d'analyse - détails

Analyse lexicale et syntaxique :

- Validation par rapport à la syntaxe SQL
- Vérification des types
  - présence des attributs et relations dans le schéma
  - compatibilité des types dans les prédicats
- Décomposition en requêtes/sous-requêtes

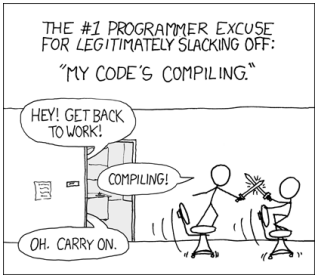
Étape de compilation



Étape de compilation - détails

Règles de passage d'une requête SQL en AR :

- SELECT pour définir une ??**projection**
- FROM pour définir les ?? **relations** (feuilles de l'arbre) et ??**jointures / produits cartésiens**
- WHERE pour définir :
  - avec les comparaisons *attribut/valeur* des ??**sélections**
  - avec les comparaisons *attribut/attribut* des ??**jointures**



Étape de compilation - exemple

SELECT R.a
FROM R, S
WHERE R.b = S.b
AND S.c = 99 ;

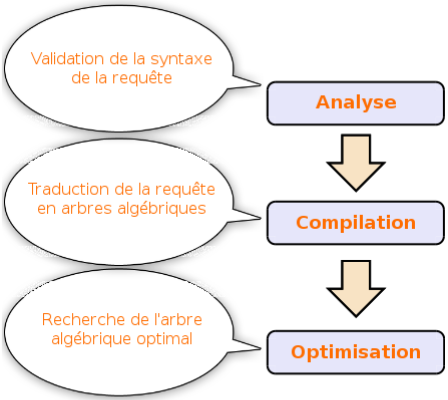
Compilation "naïve" :

- $\pi_a(\sigma_{R.b=S.b \wedge c=99}(R \times S))$

Autres possibilités :

- $\pi_a(\sigma_{R.b=S.b}(R \times \sigma_{c=99}(S)))$
- $\pi_a(R \bowtie_{R.b=S.b} (\sigma_{c=99}(S)))$

Étape d'optimisation



Étape d'optimisation - définition

Optimiser, c'est trouver le plan d'exécution d'une requête dont le coût soit minimal (i.e., le temps de réponse à la requête soit le plus rapide possible)

Qu'est-ce qui peut être optimisé dynamiquement ?

- Les accès disques

Il est indispensable que le temps lié à l'optimisation soit négligeable par rapport au temps imparti à l'exécution

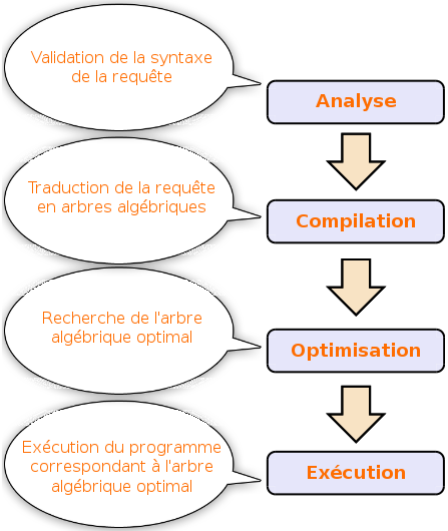
Étape d'optimisation - détails

Intuitions pour optimiser et sélectionner le plan optimal :

- Des compositions de projections et/ou sélections peuvent se réécrire en une opération de filtrage
- Réduire au plus tôt la taille et le nombre de tuples manipulés :
  - tuples moins nombreux : grâce à la ??**sélection**
  - tuples plus petits : grâce à la ??**projection**
- Réordonner les jointures, en fonction de la taille des relations manipulées
- Choisir des algorithmes efficaces pour chaque opérateur (e.g., *merge join*, *hash join* pour la jointure)

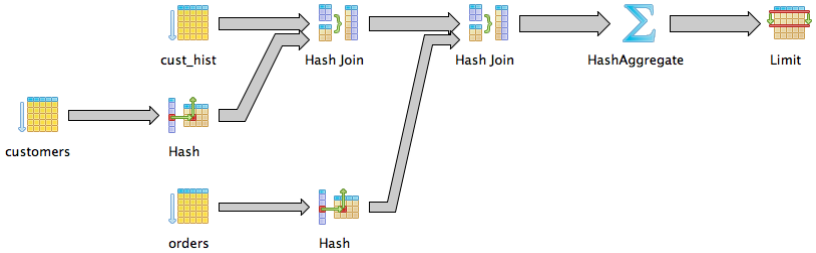
Explications détaillées dans les parties suivantes

Étape d'exécution



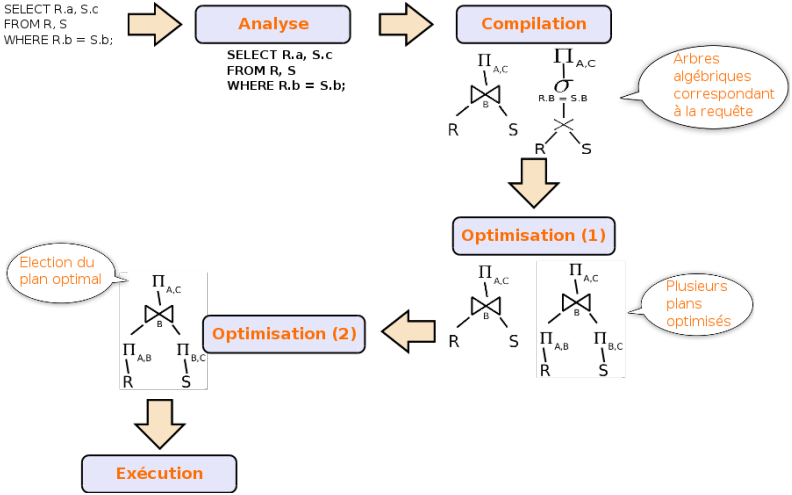
Étape d'exécution - détails

Application des algorithmes pour chaque opération du plan d'exécution sélectionné



http://mariadb.com/kb/en/mariadb/analyze-and-explain-statements/

Exemple complet de traitement d'une requête



En résumé

Traitement d'une requête :

- Validation
- Compilation en arbre algébrique
- Optimisation (génération de plans optimisés et sélection du plan optimal)
- Exécution du plan optimal

http://vidberg.blog.lemonde.fr/

Plan

Optimisation

L'objectif est de trouver une stratégie d'évaluation permettant d'accélérer l'évaluation :

- Par manipulations algébriques, indépendante de la manière dont sont stockées les informations
- En utilisant une stratégie dépendante du stockage (clés, index)

Dans ce cours, manipulations algébriques (à base de règles)

Plan d'exécution

Un plan d'éxécution $\approx$ arbre algébrique

Plus formellement, un **plan d'exécution** est un programme qui vise à évaluer une requête en algèbre relationnelle et qui consiste en une suite d'étapes parmi les suivantes :

- Application d'un opérateur unaire (sélection ou projection)
- Application d'une sélection puis d'une projection
- Application d'un opérateur binaire ($\times, \cup, \cap, \setminus, \bowtie$)
  - éventuellement précédé et/ou suivi d'opérateurs unaires

Un algorithme d'optimisation a pour but de trouver un bon (le meilleur si possible) plan d'exécution

Espace de recherche

Espace de recherche = l'ensemble des plans "équivalents" pour une même requête :

- Ceux qui donnent le même résultat
- Générés en appliquant des règles de transformation

Caractéristiques des plans d'un espace de recherche :

- Le coût de chaque plan est en général différent
- L'ordre des jointures est important

Si l'espace de recherche est très grand, l'optimiseur peut chercher un plan raisonnable, mais pas forcément optimal (par exemple sous PostgreSQL, dès qu'il y a plus de 12 jointures)

Exemple d'espace de recherche

ÉLÈVE (idE, *nomE*, *moyenneLycee*, *effectifLycee*)
CANDIDATURE (#*idE*, #*nomU*, département, *décision*)
UNIVERSITÉ (nomU, *ville*, *effectif*)

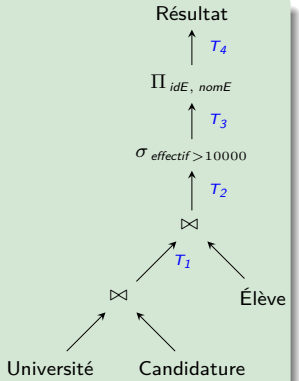
L'identifiant et le nom des élèves qui ont candidaté dans des universités avec un effectif supérieur à 10000

```
SELECT DISTINCT idE, nomE
FROM Élève e INNER JOIN Candidature c
     ON e.idE = c.idE INNER JOIN
     Université u ON c.nomU = u.nomU
WHERE effectif > 10000 ;
```

Exemple d'espace de recherche - plan 1

SELECT DISTINCT idE, nomE
FROM Élève e **INNER JOIN** Candidature c
ON e.idE = c.idE **INNER JOIN**
Université u **ON** c.nomU = u.nomU
WHERE effectif > 10000 ;

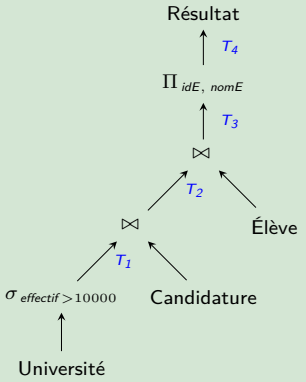
- $T_1 \leftarrow$ Joindre la table UNIVERSITÉ avec la table CANDIDATURE
- $T_2 \leftarrow$ Joindre T_1 avec la table ÉLÈVE
- $T_3 \leftarrow$ Lire T_2 et sélectionner les tuples d'*effectif* > 10000
- $T_4 \leftarrow$ Projeter T_3 sur *idE*, *nomE*



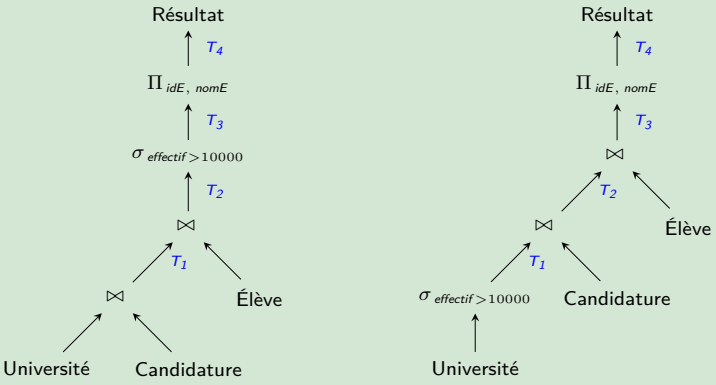
Exemple d'espace de recherche - plan 2

SELECT DISTINCT idE, nomE
FROM Élève e **INNER JOIN** Candidature c
ON e.idE = c.idE **INNER JOIN**
Université u **ON** c.nomU = u.nomU
WHERE effectif > 10000 ;

- $T_1 \leftarrow$ Lire la table ÉLÈVE et sélectionner les tuples d'*effectif* > 10000
- $T_2 \leftarrow$ Joindre T_1 avec la table CANDIDATURE
- $T_3 \leftarrow$ Joindre T_2 avec la table ÉLÈVE
- $T_4 \leftarrow$ Projeter T_3 sur *idE*, *nomE*



Exemple d'espace de recherche - choix de plan



Espace de recherche contenant deux plans d'exécution. Quel est le meilleur plan (sans considérer d'autres informations) ?

Optimisation à base de règles

Application de règles pour transformer une expression de l'algèbre relationnelle en plan d'exécution optimisé :

- Entrée : arbre représentant l'expression à évaluer
- Sortie : plan d'exécution pour la requête

L'arbre passé en entrée est construit comme suit :

- Les noeuds internes de l'arbre sont les opérateurs de l'expression
- Chaque noeud a pour fils le ou les sous-arbres construits à partir de son ou ses arguments
- Les feuilles sont des relations de la bases de données

Règles de transformation

Idées :

- Effectuer les sélections le plus tôt possible
- Combiner les sélections et les produits cartésiens pour faire des jointures
- Combiner les séquences d'opérations unaires, comme les sélections et les projections
- Chercher les sous-expressions communes dans une expression

Pour cela, on exploite des identités remarquables de l'algèbre relationnelle

Lois sur les jointures et les produits

Commutativité :

$$E_1 \bowtie_C E_2 \equiv E_2 \bowtie_C E_1 \quad (1)$$
$$E_1 \bowtie E_2 \equiv E2 \bowtie E_1 \quad (2)$$
$$E_1 \times E_2 \equiv E_2 \times E_1 \quad (3)$$

Associativité :

$$E_1 \bowtie_C (E_2 \bowtie_D E_3) \equiv (E_1 \bowtie_C E_2) \bowtie_D E_3 \quad (4)$$
$$E_1 \bowtie (E_2 \bowtie E_3) \equiv (E_1 \bowtie E_2) \bowtie E_3 \quad (5)$$
$$E_1 \times (E_2 \times E_3) \equiv (E_1 \times E_2) \times E_3 \quad (6)$$

Lois sur les projections et les sélections

Cascade de projections :

$$\pi_{A_1,\ldots,A_n}(\pi_{B_1,\ldots,B_k}(E)) \equiv \pi_{A_1,\ldots,A_n}(E) \quad (7)$$

Cascade de sélections :

$$\sigma_C(\sigma_D(E)) \equiv \sigma_{C\wedge D}(E) \quad (8)$$
$$\sigma_C(\sigma_D(E)) \equiv \sigma_D(\sigma_C(E)) \quad (9)$$

Permutations de projections et sélections :

- Si C ne porte que sur des attributs parmi $A_1, \ldots, A_n$:

$$\pi_{A_1,\ldots,A_n}(\sigma_C(E)) \equiv \sigma_C(\pi_{A_1,\ldots,A_n}(E)) \quad (10)$$

Lois sur les sélections et les autres opérations

Sélections et produits cartésiens/jointures :

- Si C ne porte que sur les attributs de E_1 :

$$\sigma_C(E_1 \times E_2) \equiv \sigma_C(E_1) \times E_2 \quad (11)$$
$$\sigma_{C \wedge D}(E_1 \times E_2) \equiv \sigma_D(\sigma_C(E_1) \times E_2) \quad (12)$$

- Si C ne porte que sur les attributs de E_1 et D sur les attributs de E_2 :

$$\sigma_{C \wedge D}(E_1 \times E_2) \equiv \sigma_C(E_1) \times \sigma_D(E_2) \quad (13)$$

Sélections et union / différence :

$$\sigma_C(E_1 \cup E_2) \equiv \sigma_C(E_1) \cup \sigma_C(E_2) \quad (14)$$
$$\sigma_C(E_1 - E_2) \equiv \sigma_C(E_1) - \sigma_C(E_2) \quad (15)$$

Projection et autres opérations

Projection et autres opérations :

- Produit cartésien :

$$\pi_{A_1,\ldots,A_i,\ldots,A_n}(E_1 \times E_2) \equiv \pi_{A_1,\ldots,A_i}(E_1) \times \pi_{A_{i+1},\ldots,A_n}(E_2) \quad (16)$$

- Union :

$$\pi_{A_1,\ldots,A_n}(E_1 \cup E_2) \equiv \pi_{A_1,\ldots,A_n}(E_1) \cup \pi_{A_1,\ldots,A_n}(E_2) \quad (17)$$

Application des règles

1. Utiliser (8) pour transformer les sélections $\sigma_{C_1 \wedge \ldots \wedge C_n}(E)$ en cascades $\sigma_{C_1}(\ldots \sigma_{C_n}(E))$

2. Pour chaque sélection, utiliser (9), (10), (11), (12), (13), (14) et (15) pour pousser les sélections en bas de l'arbre

3. Pour chaque projection, utiliser les règles (7), (16), (17) et (10) pour pousser la projection au plus bas dans l'arbre. Éliminer une projection qui conserve tous les attributs

4. Utiliser les règles (7), (8) et (10) pour combiner les cascades de sélections et projections en une sélection unique, une projection unique, ou une sélection suivie d'une projection

Application des règles (2)

5. Partitionner l'arbre résultant en groupes comme suit :
   - Créer un groupe par noeud binaire ($\times, \cup, \cap, \setminus, \bowtie$)
   - Le groupe inclut tous les ancêtres unaires intermédiaires
   - Le groupe inclut toute branche étiquetée par des opérateurs unaires et se terminant à une feuille (table)

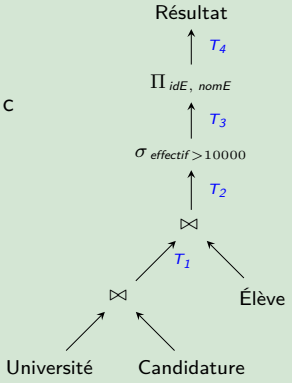
6. Chaque groupe correspond à une étape. Les étapes sont à évaluer dans n'importe quel ordre tant qu'un groupe est évalué après ses descendants

Exemple d'application des règles

L'identifiant et le nom des élèves qui ont candidaté dans des universités avec un effectif supérieur à 10000

SELECT DISTINCT idE, nomE
FROM Élève e **INNER JOIN** Candidature c
    ON e.idE = c.idE **INNER JOIN**
    Université u **ON** c.nomU = u.nomU
WHERE effectif > 10000 ;

⇒ Plan non optimisé !

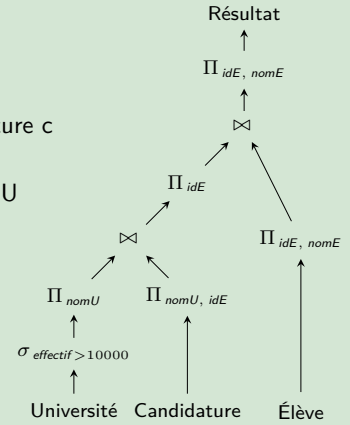


Exemple d'application des règles (2)

L'identifiant et le nom des élèves qui ont candidaté dans des universités avec un effectif supérieur à 10000

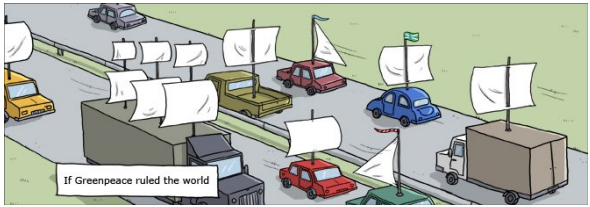
SELECT DISTINCT idE, nomE
FROM Élève e **INNER JOIN** Candidature c
ON e.idE = c.idE **INNER JOIN**
Université u **ON** c.nomU = u.nomU
WHERE effectif > 10000 ;

⇒ Plan optimisé par les règles : (11), (10), (16)



En résumé

- Optimisation à base de règles = application de règles de transformation entre les opérateurs de l'AR
- Une quinzaine de règles et un algorithme à suivre
- Idée générale : placer les sélections et les projections en bas de l'arbre (le plus tôt possible)



http://wumo.com/