Bases de données

Polytech Paris-Sud

Apprentis 4ème année

Cours 5 : Optimisation des requêtes

kn@lri.fr
http://www.lri.fr/~kn

Plan

Principe d'évaluation d'une requête

1. *Parsing* de la requête
2. Traduction en arbre d'opérateurs de l'algèbre relationnelle (π, σ, ⋈, …)
3. Optimisation :
   1. Génération de plans d'évaluation (en réordonnant les opérations élémentaires)
   2. Estimation du coût de chacun des plans (en fonction du coût des opérations élémentaires)
   3. Choix du plan le plus efficace
4. Évaluation du plan choisi
5. (Éventuellement mise à jour des statistiques)

On va voir comment optimiser l'évaluation d'une requête

Exemple pour la suite du cours

```
Sailors(mid: integer, sname: string, rating: integer, age: real);
Reserves(sid: integer, bid: integer, day: date, rname: string);
Boats(bid: integer, bname: string, capacity: integer);
```

- Sailors: 50 octets/enr., 80 enr/page, 500 pages
- Reserves: 40 octets/enr., 100 enr/page, 1000 pages
- Boats: non utilisé dans la suite

Généralités

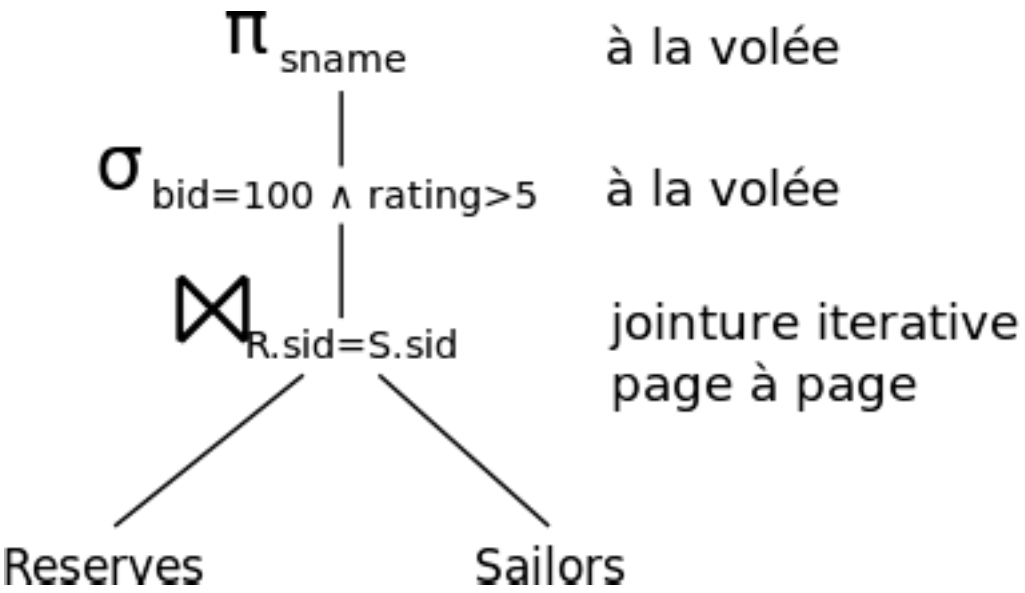
Un plan d'exécution de requête est un arbre dont les noeuds sont des opérateurs de l'algèbre relationnelle annotés avec un algorithme particulier.

- Pour une requête donnée, quels plans doit-on considérer ?
- Comment peut on estimer le coût total d'un plan

Idéalement, on veut trouver le meilleur plan. En pratique, on choisira le moins pire!

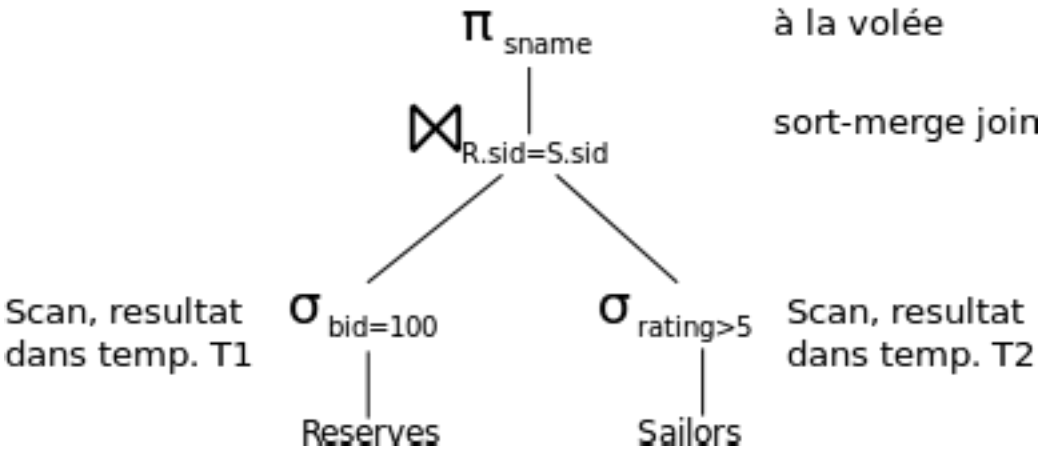
Exemple

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid = S.sid AND bid = 100 AND rating > 5
```



- Cout: 500+500*1000 E/S (pourquoi ?)
- Plusieurs occasions manquées : pas d'utilisation d'index, on aurait pu pousser la selection sous la jointure, ...
- Notre but est de trouver des plans d'exécution plus efficaces qui calculent le même résultat

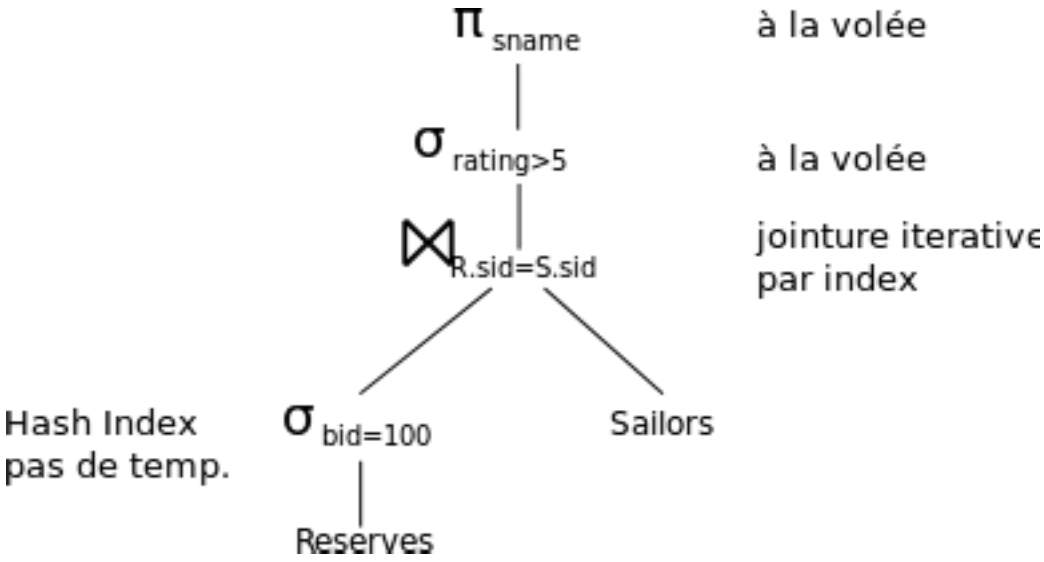
Plan alternatif 1 (sans index)



On pousse la sélection sous la jointure (car selection AND). On suppose qu'on a 100 bateaux, 10 notes et distributions uniformes.

- Scan Reserves (1000 E/S) et écriture de 10 pages dans T1
- Scan Sailors (500 E/S) et écriture de 250 pages dans T2
- Tri T1 (3*10 E/S), Tri T2 (8*250 E/S), fusion (10+250 E/S)
- Total: 4050 E/S
- Si on pousse la projection, T1 ne contient que sid, T2 uniquement sid et sname (cout < 2000 E/S)

Plan alternatif 2 (avec index)



On suppose un hash-index groupant sur `bid`

- Accès au premier enregistrement `bid=100` 1.2 E/S
- Accès aux suivants: 9 E/S
- `sid` est une clé pour Sailors. On a un hash-index dessus (forcément non-groupant)
- Pour chacun des 10 * 100 enr. tels que `bid=100` on cherche l'enregistrement de Sailors avec le même `sid` (1.2 E/S/enr)
- Coût total: 1000 * 1.2 + 10.2 = 1210 E/S

Algorithme général de choix de plan

- Cas mono-relation (une seule table dans le `FROM`):
  1. On énumère tous les plans (en tenant compte des équivalences de l'algèbre relationnelle)
  2. On calcule le coût de chaque plan
  3. On choisit le plan de moindre coût
- Cas multi-relations (plusieurs tables dans le `FROM`):
  1. Trop de plan pour les énumérer tous, on choisit des arbres ayant une certaine forme
  2. On calcule le coût de chaque plan
  3. On choisit le plan de moindre coût
- On sait (cours 4) estimer le coût d'un opérateur en fonction de la taille de l'entrée. On va enchaîner les opérateurs donc il faut estimer la taille du résultat pour calculer le coût de l'opérateur suivant!

Plan

Statistiques et catalogues

On a besoin d'informations numériques sur les relations et les indexes. Un catalogue contient en général:

- Le nombre d'enregistrements (`NEnr`) et le nombre de pages (`NPages`) de la relation.
- Le nombre de clés distinctes (`NClés`) pour les indexes ainsi que leur taille en pages
- La hauteur ainsi que les clés min/max dans l'index, pour les arbres

Les catalogues sont mis à jours périodiquement mais pas à chaque mise à jours, pour ne pas impacter les performances.

Estimation du nombre de résultats et facteur de réduction

On considère une requête de la forme:

SELECT *attributs* FROM *tables* WHERE e_1 AND … AND e_n

- La taille maximale TMax du résultat est le produit des tailles des tables se trouvant dans le FROM
- Le facteur de réduction de chaque expression e caractérise l'impact de ce terme sur la taille du résultat
- La taille finale du résultat est approximée par: $TMax * RF_1 * … * RF_n$

On fait la supposition que les expressions sont indépendantes.
Exemples de facteurs de réduction:

- att = valeur : 1 / NClés si att este une clé pour un index I
- $att_1 = att_2$: 1/Max(NClé(I1), NClé(I2)) (avec att_i une clé de Ii)
- att > valeur : (Max(I)-valeur)/(Max(I) - Min(I))

Équivalences de l'algèbre relationnelle

Permet de réordonner les jointures et de « pousser » les sélections et les projections sous les jointures

- Sélections :
  - $\sigma_{c_1 \wedge \ldots \wedge c_n}(R) \equiv \sigma_{c_1}(\ldots (\sigma_{c_n}(R)))$ [Cascade]
  - $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$ [Commutativité]
- Projections :
  - $\pi_{a_1, \ldots, a_n}(\ldots(\pi_{z_1, \ldots, z_m}(R)) \equiv \pi_{a_1, \ldots, a_n}(R)$ [Cascade]
- Jointures :
  - $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$ [Associativité]
  - $(R \bowtie S) \equiv (S \bowtie R)$ [Commutativité]

Autres équivalences

- Une projection commute avec une selection qui utilise uniquement les attributs de la projection
- Une selection entre des attributs de deux arguments d'un produit cartésien peut être converti en jointure: $\sigma_{\varphi}(R \times S) \equiv R \bowtie_{\varphi} S$
- Une selection sur des attributs de R commute avec la jointure $R \bowtie S$ (c'est à dire: $\sigma(R \bowtie S) \equiv \sigma(R) \bowtie S$)
- Règle similaire pour pousser les projections sous jointure

Plan

Modèle de calcul

Les SGBD modernes utilisent un modèle de calcul *pull*. L'opérateur le plus « haut » (racine) dans l'arbre de requête « tire » (*pull*) le résultat de ses sous-arbres (similaire à l'appel de *next* sur les iterateurs de la bibliothèque standard Java). Cela permet de *pipeliner* les opérateurs. Certains opérateurs « bloquent » le *pipeline* (en particulier les tris et aggrégats).

Cas mono-relation

Dans le cas mono-relation (i.e. sans jointure), la requête est composée forcément de selections, projections et aggrégats (`max`, `count`, `average`, ...)

1. Pour chaque sous-terme, on considère tous les accès possibles (scan, utilisation de l'index, ...) et on prend le moins coûteux
2. Les opérateurs restants sont calculé à la volée (en pipelinant les opérations)

Estimation du coût pour les plans mono-relation

- Si on a un index I pour une selection sur clé primaire : `Hauteur(I) + 1` pour un arbre B+, `1.2` pour un hash-index
- Si on a un index I groupant pour plusieurs selection σ_1, ..., σ_n :

`(NPages(I) + NPages(R))* RF(`σ_1`) * … * RF(`σ_n`)`

- Si on a un index I non-groupant pour plusieurs selection σ_1, ..., σ_n :

`(NPages(I) + NEnr(R))* RF(`σ_1`) * … * RF(`σ_n`)`

- Scan séquentiel à `R`: `NPages(R)`

Exemple de calcul de coût

```
SELECT S.sid FROM Sailors S WHERE S.rating = 8;
```

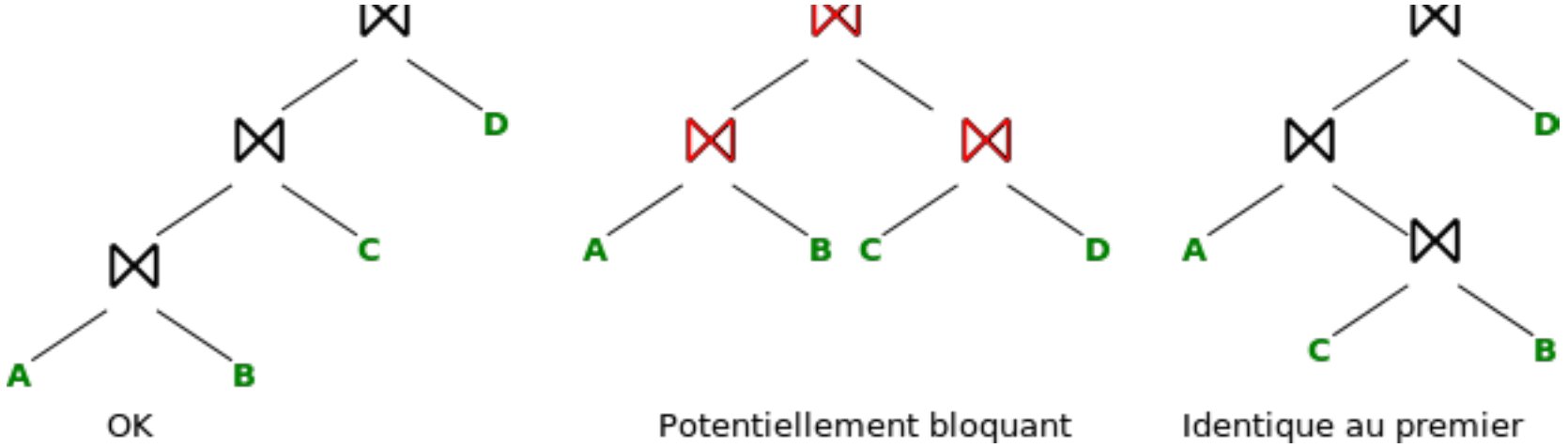
$\pi_{sid}(\sigma_{rating = 8}(R))$

- Avec un index sur `rating`:
  - Groupant: `1/NClés(I) * (NPages(I) + NPages(R))`. Avec des valeurs numériques: `1/10 * (50+500) = 55 E/S`
  - Non-groupant: `1/NClés(I) * (NPages(I) + NEnr(R))`. Avec des valeurs numériques: `1/10 * (50+40000) = 4005 E/S`
- Scan : on récupère toutes les pages et on filtre: `500` E/S

Note: Une fois que l'on a sélectionné un enregistrement, la projection est « gratuite » (en terme d'E/S) car le résultat n'a pas à être sauvé dans une table temporaire

Requêtes multi-relations

- Les choix vont être guidé par les jointures
- Si on considère uniquement n jointures (pas de projections ni de selections dans le plan de requête). Le nombre de plans possible est le nombre d'arbre binaires ayant n noeuds internes (exponentiel en n, exactement: nombre de Catalan d'indice n). Beaucoup trop pour les énumérer tous.
- On se restreint aux arbres gauches en profondeur qui permettent d'énumérer tous les plans complètement « *pipelinable* »



Ce ne sont que des heuristiques pour réduire l'espace de recherche, on n'est pas sur d'avoir la solution optimale!

Énumération des plans gauches en profondeur 1/2

Toujours exponentiel (mais moins)

Tous les arbres différent maintenant dans l'ordre dans lequel on fait les jointures, la méthode d'accès pour chaque relation et les algorithmes de jointure utilisés

On applique l'heuristique suivante:

1. 1ère passe: on trouve la meilleure manière de calculer chaque relation individuellement
2. 2ème passe: on trouve la meilleure manière de joindre deux à deux les résultats de la passe 1
3. nème passe: on trouve la meilleure manière de joindre deux à deux les résultats de la passe (n-1)

Comment sélectionner les « meilleurs » jointures ? On garde pour chaque ordre de résultat intermédiaire celle de moindre coût

Énumération des plans gauches en profondeur 2/2

Exemple: si on a la possibilité de faire une jointure itérative de cout 1000, une jointure par hash de coût 500 et une jointure sort-merge de coût 1500, on garde les version hash et sort-merge (car il est possible que le fait d'avoir les résultats déjà trié rendent le coût moindre à l'étape suivante)

On garde les `ORDER BY`, `GROUP BY`, aggrégats, ... pour la fin, en profitant si possible des ordres des résultats faits par les jointures précédentes

On évite tant que possible de faire des produits cartésiens (en poussant les selections par exemple)

Requêtes imbriquées

```
SELECT … FROM … WHERE
 … e AND EXISTS  (SELECT … WHERE … FROM …)
```

On optimise d'abord la requête la plus « interne »

On optimise ensuite la requête englobante en utilisant prenant en compte le coût de la requête interne pour chaque « évaluation » du `WHERE`