Bases de données

Polytech Paris-Sud

Apprentis 4ème année

Cours 5 : Optimisation des requêtes

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

Plan

1 Rappels ✔
2 Stockage ✔
3 Indexation ✔
4 Optimisation des opérateurs ✔
5 Optimisation de requêtes
- 5.1 Motivation et introduction
- 5.2 Estimation de coût
- 5.3 Énumération de plans

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(sid: 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: 20 octets/enr., 200 enr/page, 200 pages

Généralités

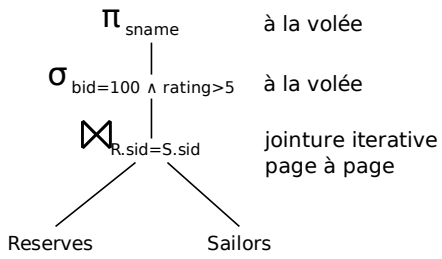
Un **plan d'exécution** de requête est un **arbre** dont les noeuds sont des opérateurs de l'algèbre relationnelle annotés avec un algorithme particulier.

- 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!

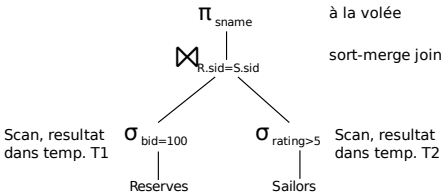
Exemple

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



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

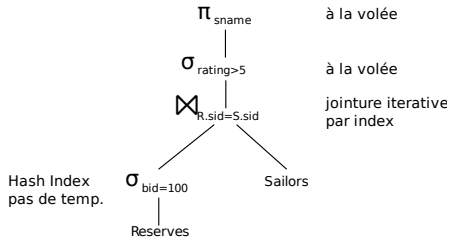
Plan alternatif 1 (sans index)



On pousse la sélection sous la jointure (car sélection `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

1 Rappels ✔
2 Stockage ✔
3 Indexation ✔
4 Optimisation des opérateurs ✔
5 Optimisation de requêtes
  5.1 Motivation et introduction ✔
  5.2 Estimation de coût
  5.3 Énumération de plans

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: $\texttt{TMax} * RF_1 * \ldots * RF_n$

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
- $\texttt{att}_1 = \texttt{att}_2$: `1/Max(NClé(I1), NClé(I2))` (avec $\texttt{att}_i$ 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 sélection qui utilise uniquement les attributs de la projection
- Une sélection 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 sélection 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

1 Rappels ✔
2 Stockage ✔
3 Indexation ✔
4 Optimisation des opérateurs ✔
5 Optimisation de requêtes
  5.1 Motivation et introduction ✔
  5.2 Estimation de coût ✔
  5.3 Énumération de plans

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 itérateurs de la bibliothèque standard Java). Cela permet de ***pipeliner*** les opérateurs. Certains opérateurs « **bloquent** » le *pipeline* (en particulier les tris et agrégats).

Cas mono-relation

Dans le cas mono-relation (i.e. sans jointure), la requête est composée forcément de sélections, projections et agrégats (`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 sélection 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 sélection σ_1, ..., σ_n : `(NPages(I) + NPages(R))* RF(`σ_1`) * … * RF(`σ_n`)`
- Si on a un index I non-groupant pour plusieurs sélection 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;
```

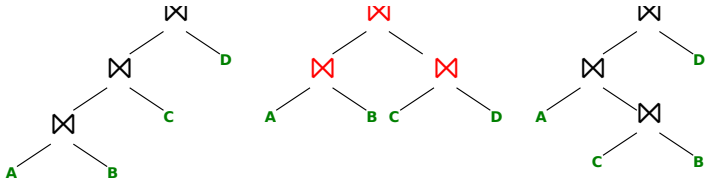
$\pi_{sid}(\sigma_{rating = 8}(S))$

- 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 sélections 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 sûr 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 « meilleures » jointures ? On garde pour chaque ordre de résultat intermédiaire celle de moindre coût

Énumération des plans gauches en profondeur 2/2

Points à prendre en compte pour le calcul du coût d'un plan

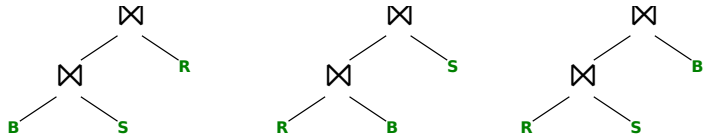
1. Éviter les plans qui génèrent des produits cartésiens, sauf si c'est indispensable
2. Les projections, sélections, et jointures itératives par index peuvent êtres faites en *pipeline* (ou *streaming* sans itération/matérialisation des résultats intermédiaires
3. Cela peut valoir le coup d'utiliser un *merge-sort* join (possiblement coûteux) si on demande les résultats dans un certain ordre (`ORDER BY` compatible avec celui de la jointure). Cela évite de faire une jointure suivie d'un tri.
4. Pousser les sélections/projections plus bas dans le plan permet de faire diminuer la taille des résultats intermédiaires. Attention cependant, appliquer une sélection ou une projection à une table T munie d'un index crée une table T' plus petite mais *sans index*.

Exemple

Considérons la requête :

```
SELECT sname, bname FROM Boats B, Sailors S, Reserves R WHERE
B.bid = R.bid AND S.sid = R.sid AND S.rating > 8
```

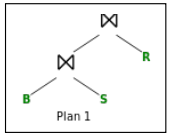
Deux jointures ($B \bowtie R$ et $R \bowtie S$) et une sélection sur S. On suppose un hash-index sur S.sid et un hash-index sur B.bid, les valeurs de notes vont de 1 à 10, uniformément réparties. Tous les sid et tous les bid sont présents dans R. Les hash-index sont non-groupants et de coût d'accès 2. Quels sont les plans possibles ?



Exemple (Calculs préliminaires)

- Taille d'une page : 4000 octets
- Taille de S : 500 pages, 40 000 enr.
- Taille de R : 1000 pages, 100 000 enr.
- Taille de B : 200 pages, 40 000 enr.
- Taux de sélectivité de `rating > 8` : 20%
- $\sigma_{rating>8}(S)$: 8000 enr. ou 100 pages

Exemple (Plan 1)



On choisi de faire d'abord une jointure entre B et S (i.e. un produit cartésien car il n'y a pas de condition de jointure entre ces deux tables). Puis la jointure de la table résultante avec B sur les attributs (bid,sid).

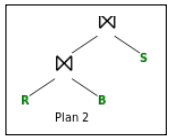
Manière la plus efficace de calculer S : appliquer la sélection directement sur S, puis la jointure $B \bowtie (\sigma(S))$. Pas d'utilisation d'index possible, jointure page à page (car on génère tout le produit cartésien) : `100 × (100 + 200) = 30 000` E/S (pages) ou 19 200 000 enr.

Puis jointure itérative page à page avec R (pas d'index sur R, pas d'index sur le résultat précédent qu'on vient de créer en mémoire): `1 000 × (30 000 + 1000) = 31 000 000 E/S.`

Coût total: `31 300 000 E/S` (les projections sont faites en pipline à la fin)

Plan complètement inefficace. On ignorera les plans contenant un produit cartésien, sauf si c'est le seul moyen de calculer la requête (`SELECT * FROM A, B`).

Exemple (Plan 2 v1)



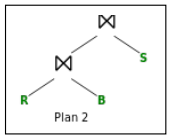
On choisi de faire d'abord une jointure entre R et B, sur l'attribut `bid` puis jointure du résultat intermédiaire sur `sid`.

On dispose d'un index sur `B.bid`. On effectue une jointure itérative par index : `1000 + 100 000 × 3 : 301 000 E/S` (2 pour le hash-index et 1 pour la lecture des données depuis l'index).

La deuxième jointure peut être faite à la volée (jointure itérative par index sur `S.sid`) et la condition de sélection testée à la volée. Coût total 100 000 × 3 (pour chacun des enregistrement précédents, on paye un accès d'index + un accès à la ligne correspondante dans S).

Coût total: `601 000 E/S` (les projections sont faites en *pipline* à la fin)

Exemple (Plan 2 v2)



On choisi de faire d'abord une jointure entre R et B, sur l'attribut `bid` puis jointure du résultat intermédiaire sur `sid`.

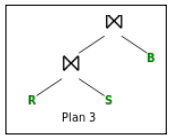
On n'utilise pas l'index sur `B.bid`. On effectue une jointure itérative page à page : `200 + 200 × 1000 : 200 200 E/S` . On a 100 000 résultats (car tous les B.bid sont présents dans la table R). Un enregistrement du résultat fait environ 40+20 = 60 octets donc 66 enregistrements par pages de 4000 octets donc 1515 pages de résultats.

On applique la sélection sur S par un scan linéaire: `500 E/S` et `100 pages` de résultats

On fait une jointure page à page des deux résultats précédents : `100 + 100 × 1515 = 100 615 E/S`.

Coût total: `102 630 E/S` (les projections sont faites en *pipline* à la fin)

Exemple (Plan 3)



On choisi de faire d'abord une jointure entre R et S, sur l'attribut `sid` puis jointure du résultat intermédiaire sur `bid`.

On applique la sélection sur S par un scan linéaire: `500 E/S` et `100 pages` de résultats

On effectue une jointure itérative page à page : `100 + 100 × 1000 : 100 100 E/S` . On a 100 000 résultats (car tous les S.sid sont présents dans la table R, même après sélection car distribution uniforme). Un enregistrement du résultat fait environ 40+50 = 90 octets donc 44 enregistrements par pages de 4000 octets donc 2272 pages de résultats.

On fait une jointure page à page des du résultat précédent avec B : `200 + 200 × 2272 = 454 600 E/S`.

Coût total : 555 200 E/S

Exemple (conclusion)

- Utiliser l'index n'est pas toujours payant, surtout s'il est non-groupant, car on ajoute un facteur qui est le nombre de résultats, pas le nombre de pages
- On a fait certaines approximations « à la louche » (taille des enregistrements résultants d'une jointure, nombre des enregistrements résultants)
- On n'a pas considéré le fait que pousser les projections plus bas pour ne garder que les colonnes strictement nécessaires pouvait faire baisser la taille des tables intermédiaires

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`