SGBD-R : traitement de requêtes

Algorithmes de tri et de jointure

Guillaume Raschia — Nantes Université

originaux de Philippe Rigaux, CNAM

Dernière mise-à-jour : 20 janvier 2023

Plan de la session

Le tri externe (S6.3)

La jointure (S6.4)

Le tri externe (S6.3)

Rappel : le tri

Opération qui ordonne un ensemble d'éléments sur une clé de tri.

Les algorithmes

- version naïve : n parcours successifs pour trouver le minimum à chaque itération ($O(n^2)$)
- les algorithmes de tri opèrent en mémoire centrale
- il en existe une grande variété :
  - tri rapide, en $O(n \log n)$
  - tri fusion (ou tri dichotomique)
  - tri à bulles, par tas, par sélection, par insertion, ...

Tri externe

Le tri externe est utilisé pour

- les algorithmes de jointure (*sort/merge*)
- l'élimination des doublons (clause `distinct`)
- les opérations de regroupement (`group by`)
- … et bien sûr pour les `order by`

C'est une opération qui peut être très coûteuse sur de grands jeux de données.

Coût d'une opération externe

Les données sont stockées dans des blocs (ou pages) sur disque.

Lors d'une opération :

- elles sont chargées (lues) à la demande en mémoire principale, par page ;
- elles peuvent être temporairement réécrites sur disque au cours de l'opération.

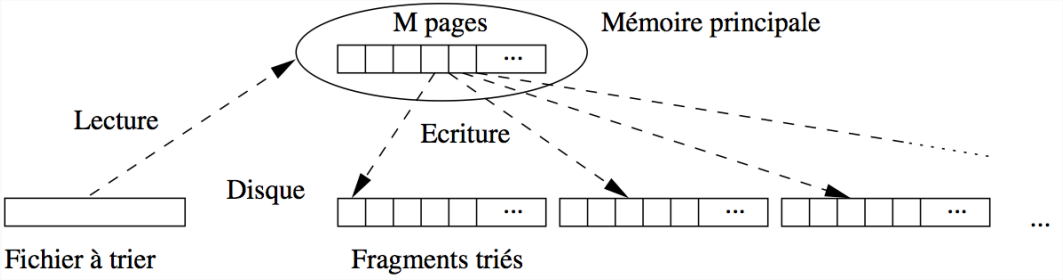
Mesure du coût

Nombre i/o de lectures et d'écritures de blocs sur disque

Le coût usuel, en nombre d'opérations élémentaires, devient négligeable !

Première phase : le tri

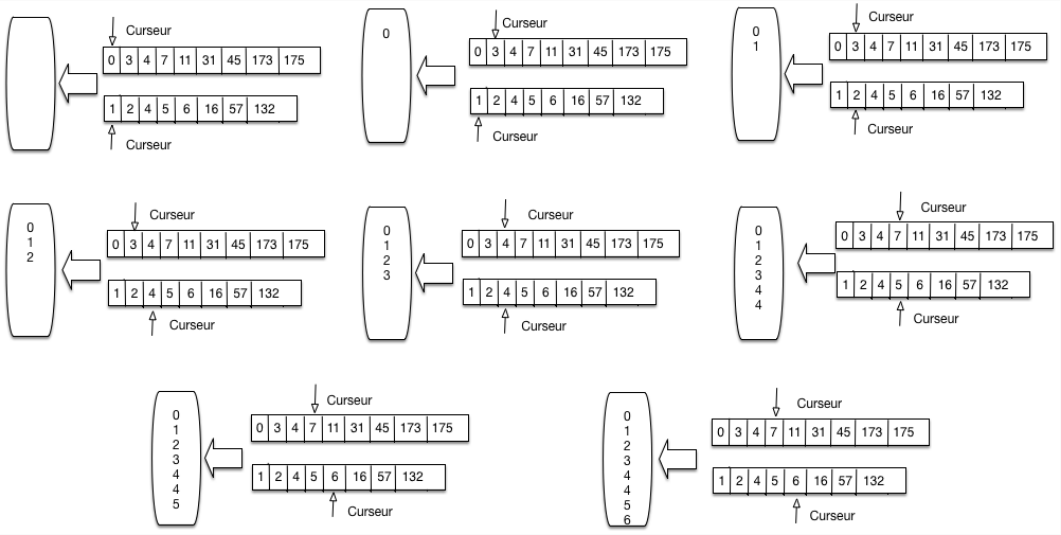
On remplit la mémoire, on trie, on vide dans des fragments, et on recommence.



Coût : une lecture + une écriture du fichier.

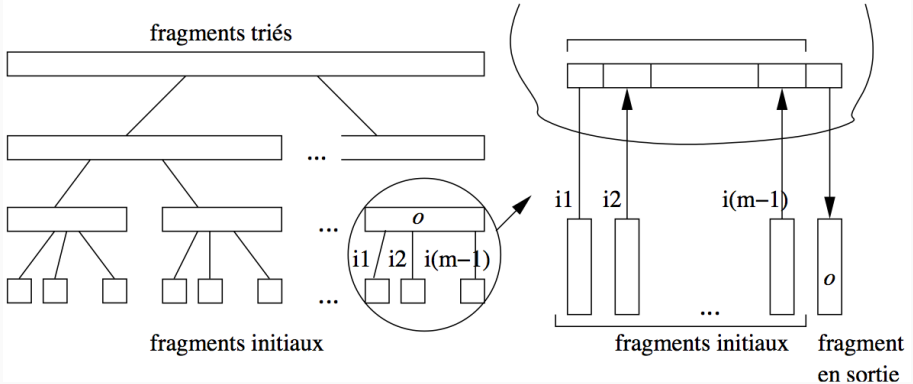
M désigne la taille de la zone tampon (mémoire principale), en nombre de blocs

Fusion de listes triées



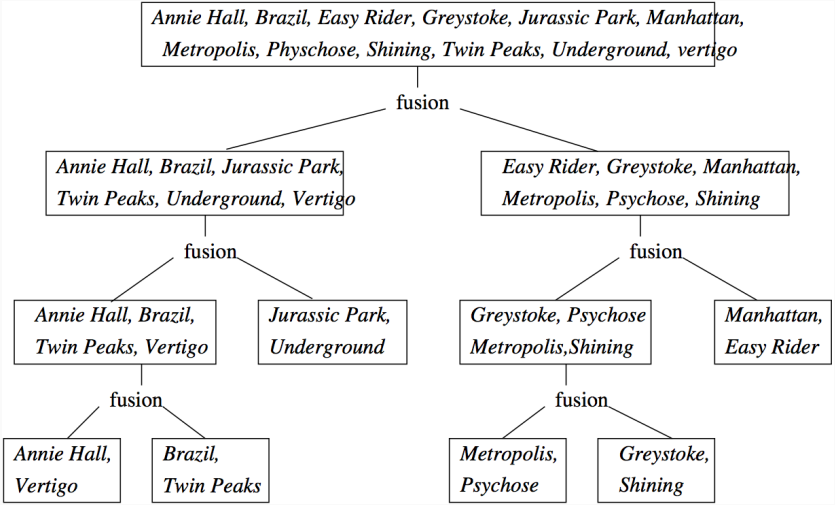
Deuxième phase : la fusion

On groupe les fragments par $M - 1$, et on fusionne.



Coût : autant de lectures/écritures du fichier que de niveaux de fusion.

Illustration avec $M = 3$



Essentiel : la taille M de la zone de tri

Un fichier de 75 000 pages de 4 Ko, soit 307 Mo.

- $M > 307$ Mo : une lecture, soit un coût de 75 Ki/o.
- $M = 2$ Mo, soit 500 pages.
  - Le tri donne $\lceil \frac{307}{2} \rceil = 154$ fragments : 1 lecture et 1 écriture des données.
  - On fait la fusion avec 154 pages (< 500 donc un seul niveau de fusion) : 1 lecture.

  Coût total de $614 + 307 = 921$ Mo, soit 230 250 i/o.

Remarque

Il faut allouer beaucoup de mémoire pour passer de 1 à 0 niveau de tri.

On ne compte jamais l'écriture du résultat de l'opération dans le coût (cf. pipeline)

Avec très peu de mémoire

$M = 1$ Mo, soit 250 pages. Rappel : 75 000 pages de 4 Ko, soit 307 Mo de données.

1. On obtient d'abord 307 fragments triés.
2. On fusionne les 249 premiers fragments, puis les 58 restant.
3. On fusionne les 2 super-fragments.

Coût total : $614 + 614 + 307 = 1\,535$ Mo, soit 383 750 i/o.

Résultat : grosse dégradation entre 2 Mo et 1 Mo.

L'opérateur de tri

L'opérateur de tri est bloquant

- Les deux phases du tri-fusion sont effectuées pendant le `open()`.
- Le `next()` ne fait que lire, un à un, les nuplets dans le résultat du tri.

Conséquence : latence importante des requêtes impliquant un tri.

Grosse différence entre :

```
select * from Film
```

et

```
select * from Film order by titre
```

La jointure (S6.4)

L'opérateur qui nous manque

La jointure : opération très courante, potentiellement coûteuse.

L'opérateur de jointure complète notre petit catalogue pour (presque) toutes les requêtes SQL conjonctives [1].

```
select a1, a2, .., an
from T1, T2, ..., Tm
where T1.x = T2.y and ...
order by ...
```

Plusieurs algorithmes possibles.

1. sans union ni négation.

Principaux algorithmes

Jointure avec index

- Algorithme de jointure par boucles imbriquées indexées
  - avec 1 ou 2 indexes.

Jointure sans index

- Le plus simple : boucles imbriquées (non indexées).
- Plus sophistiqué :
  - la jointure par hachage
  - la jointure par tri-fusion.

Jointure avec index

Très courant; on effectue naturellement la jointure sur les clés primaires/étrangères.

- Les films et leur réalisateur

```
select * from Film as f, Artiste as a
where f.id_realisateur = a.id
```

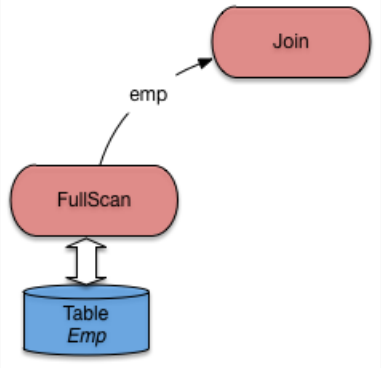
- Les employés et leur département

```
select * from emp e, dept d
where e.dnum = d.num
```

Garantit au moins un index, la clé primaire, sur la condition de jointure.

Jointure avec index : l'algorithme

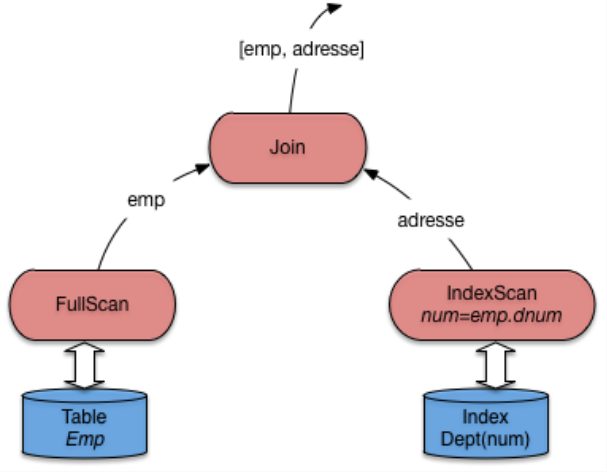
On parcourt séquentiellement la table contenant la clé étrangère.



On obtient des nuplets `employé`, avec leur no de département.

Jointure avec index : l'algorithme

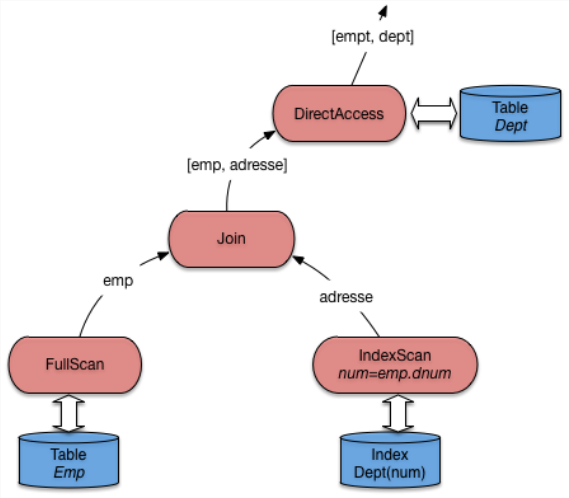
On utilise le no de département pour un accès à l'index `Dept`.



On obtient des paires `[employé, @Dept]`.

Jointure avec index : l'algorithme

Il reste à résoudre l'adresse (`@Dept`) du département avec un accès direct.



On obtient des paires `[employé, dept]`.

Jointure avec index, analyse

Avantages

- Efficace : un parcours, plus des recherches par adresse
- Favorise le temps de réponse **et** le temps d'exécution

Pour quel coût ?

Jointure $R \bowtie S$, avec T_i le nombre de nuplets et B_i le nombre de blocs de la table $i \in \{R, S\}$;

- Parcours séquentiel de la table gauche R (boucle principale) : B_R lectures
- Accès direct à chaque nuplet de S : T_R blocs lus sous quelles hypothèses ?

Avec deux indexes

Un index sur la clé de jointure dans chacune des relations R et S.

Procédure

1. Parcours des feuilles triées de chaque index (arbre B), puis
2. Fusion des listes d'adresses de nuplets de R et de S sur la condition de jointure (cf. algo. de jointure par tri-fusion), et enfin
3. Accès direct des seuls nuplets à joindre.

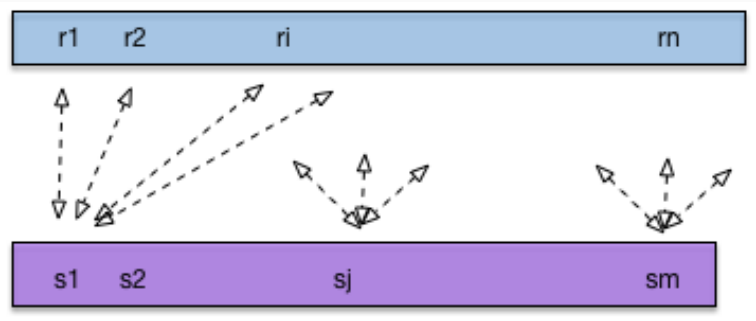
Inconvénient

La multiplication des accès directs pénalise la performance.

- Si la proportion de nuplets est grande, le balayage séquentiel est préférable.

Jointure en mémoire

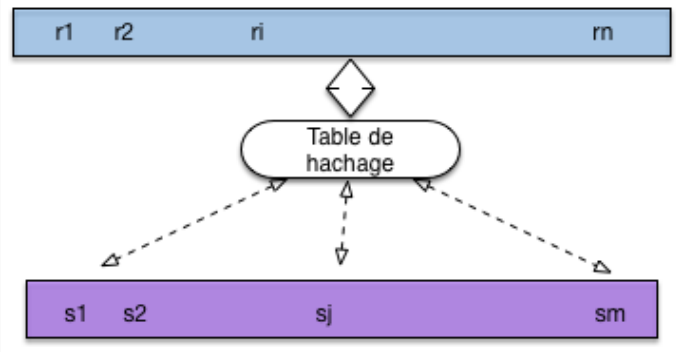
Pas d'index ? La méthode de base est d'énumérer toutes les solutions possibles.



Coût quadratique. Acceptable pour deux petites tables.
Appelons cette méthode JoinList.

Variante de JoinList

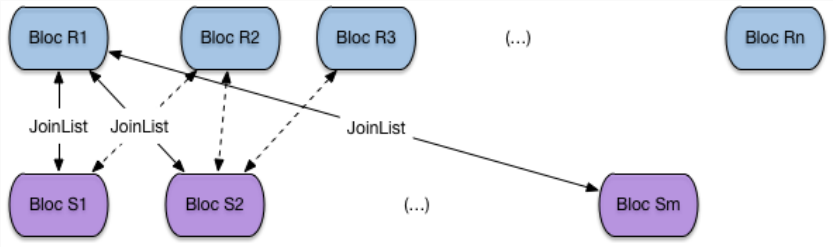
Meilleure solution : construire une table de hachage sur l'une des tables.



Evite les $O(n^2)$ comparaisons. Possible uniquement pour les équi-jointures !

Jointure par boucles imbriquées

JoinList sur chaque paire de blocs du produit cartésien.



Coût de $R \bowtie S$

- Chargement de R, par bloc : B_R i/o
- Chargement de l'intégralité de S, pour chaque bloc de R : $B_R \cdot B_S$ i/o
- Besoin de $M = 2$ blocs uniquement.

Jointure par boucles imbriquées, améliorée

Que peut-on faire de la mémoire inexploitée ($M > 2$) ?

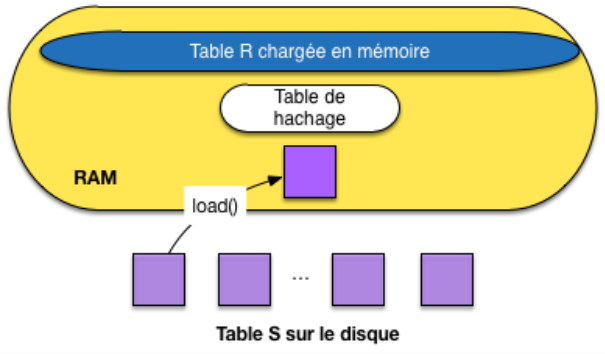
Chargement de R par lots de $M - 1$ blocs, pour saturer la zone tampon.

Coût modifié de $R \bowtie S$

- Chargement de R : B_R i/o
- Chargement de l'intégralité de S, pour chaque lot de $M - 1$ blocs de R : $\lceil \frac{B_R}{M-1} \rceil \cdot B_S$ i/o

Cas favorable de la jointure par boucles imbriquées

Une des deux tables tient intégralement en mémoire.



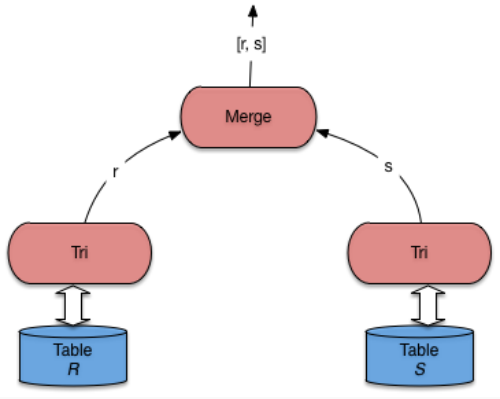
On charge l'autre bloc par bloc ; on applique JoinList.

Une seule lecture de chaque table suffit.

Dans tous les cas, on préfère la plus petite table « à gauche » (boucle externe).

Jointure par tri-fusion

Les 2 tables doivent être triées sur la clé de jointure (cf. tri-fusion externe).

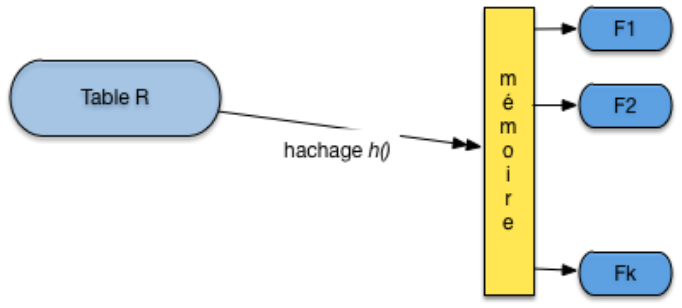


L'opérateur Merge fusionne 2 listes triées pour construire les nuplets du résultat.

Coût des tris + une lecture de chaque table pour la fusion.

Jointure par hachage

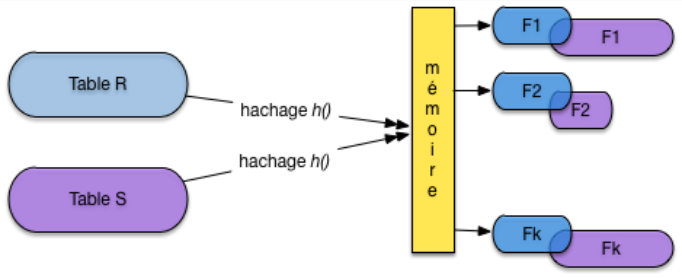
On hache la plus petite des deux tables en k fragments.



Essentiel : les fragments doivent tenir, chacun, en mémoire.

Jointure par hachage

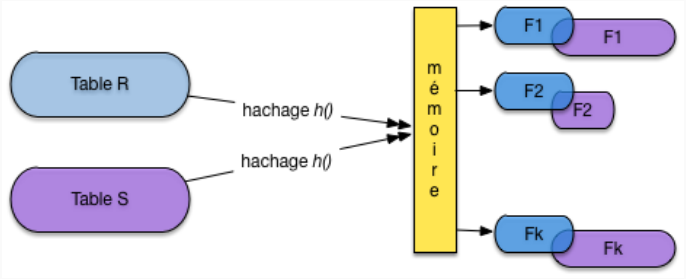
On hache la seconde table, avec la même fonction $h()$, en k autres fragments.



Cette fois, on n'impose pas la contrainte que les fragments tiennent en mémoire.

Jointure par hachage

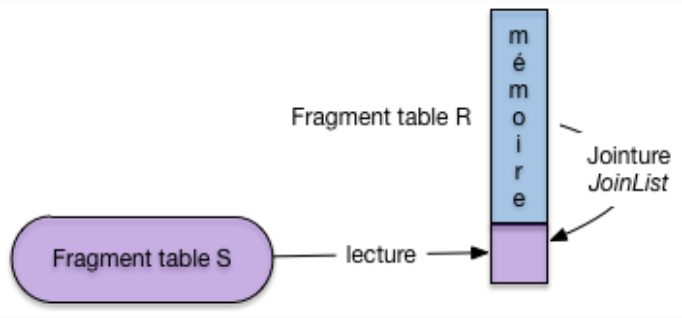
On effectue la jointure sur les paires de fragments $(F_1^R, F_1^S), (F_2^R, F_2^S), (F_k^R, F_k^S),$



Propriété : Deux nuplets r et s peuvent être joints seulement s'ils sont dans des fragments associés.

Illustration : phase de jointure

On charge F_R^i de R en mémoire ; on parcourt F_S^i de S et on joint.



Déjà vu ? Oui : jointure par boucles imbriquées quand une table tient en mémoire.

Résumé : la jointure

Un opérateur potentiellement coûteux. Quelques principes généraux :

- Si une table tient en mémoire : jointure par boucles imbriquées, ou hachage.
- Si au moins un index est utilisable : jointure par boucles imbriquées indexées.
- Si l'une des deux tables est beaucoup plus petite que l'autre : jointure par hachage.
- Sinon : jointure par tri-fusion.

Décision très complexe, prise par le système en fonction des statistiques.