Bases de données relationnelles

« *Évaluation de requête* »

G. Raschia

Date de la dernière modification : 14 septembre 2016

Dpt. INFO — Polytech Nantes

Au menu

Aperçu de la chaîne de traitement

Réécriture de requête

Estimation de taille

En route vers le plan physique

Estimation du coût I/O

Éléments de tuning pour la performance

Introduction

De quoi parle-t-on ?

Traitement de requête

$Q \rightarrow$ Plan d'exécution de la requête

Cible

Système de gestion de bases de données relationnelles

- Et pour les autres modèles ?

De quoi parle-t-on ? ...vraiment

Example

```
SELECT B, D
    FROM R, S
    WHERE R.A='c' AND
          S.E=2   AND
          R.C=S.C ;
```

De quoi parle-t-on ? ...vraiment

$\mathbf{R} =$

A	B	C
a	1	10
b	1	20
c	2	10
d	2	35
e	3	45

$\mathbf{S} =$

C	D	E
10	x	2
20	y	2
30	z	2
40	x	1
50	y	3

Réponse

B	D
2	x

Comment évalue-t-on la requête ?

Une première tentative

1. Construire le produit cartésien (`FROM`)
2. Filtrer les n-uplets (`WHERE`)
3. Projeter (`SELECT`)

Retour à l'exemple

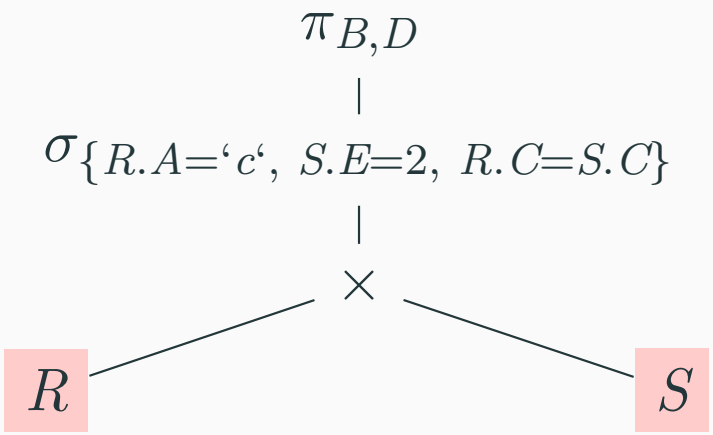
$\mathbf{R} \times \mathbf{S} =$

$R.A$	$R.B$	$R.C$	$S.C$	$S.D$	$S.E$
a	1	10	10	x	2
a	1	10	20	y	2
...	...	...	...	...	...
c	2	10	10	x	2

Algèbre relationnelle

...utilisée pour décrire des plans

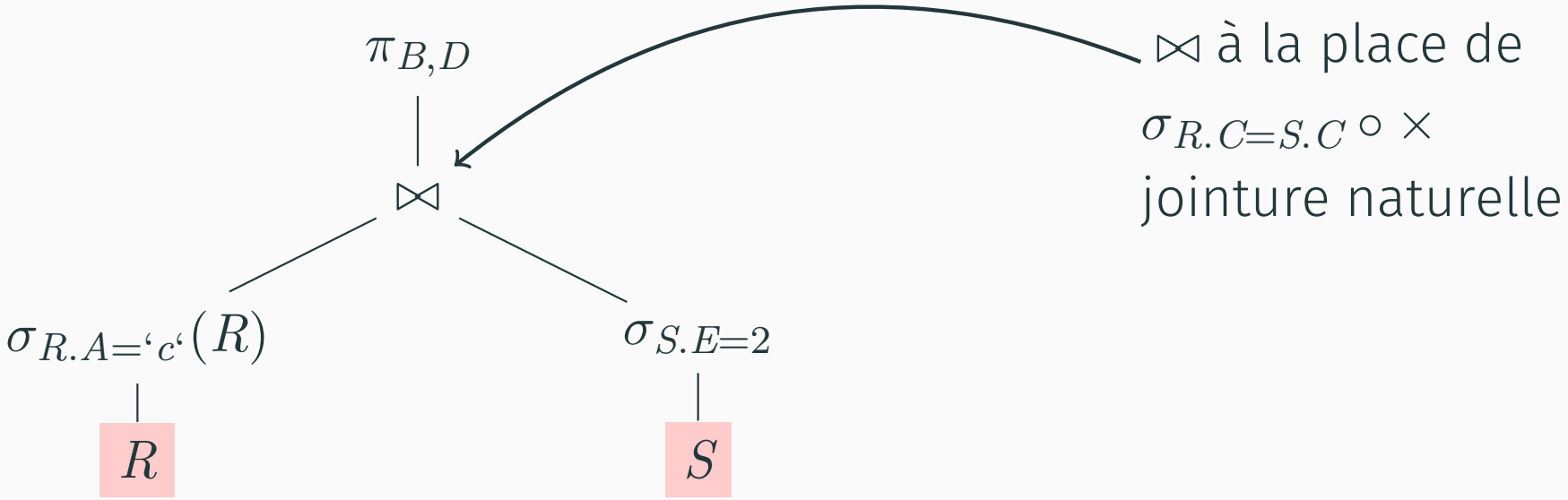
Example (Plan numéro 1)



- Exp. algébrique : $\pi_{B,D}\big(\sigma_{\{R.A=\text{`}c\text{`},\ S.E=2,\ R.C=S.C\}}(R \times S)\big)$

Une autre idée

Plan numéro 2



...et sa réalisation

$\mathbf{R}$

A	B	C
a	1	10
b	1	20
c	2	10
d	2	35
e	3	45

$\blacktriangleright$

$\sigma(\mathbf{R})$

A	B	C
c	2	10

$\sigma(\mathbf{S})$

C	D	E
10	x	2
20	y	2
30	z	2

$\blacktriangleleft$

$\mathbf{S}$

C	D	E
10	x	2
20	y	2
30	z	2
40	x	1
50	y	3

$. \bowtie . =$

A	B	C	D	E
c	2	10	x	2

$\pi_{\mathbf{B},\mathbf{D}}(.) =$

B	D
2	x

Encore une idée

Plan numéro 3

Utiliser les indexes sur $R.A$ et $S.C$

1. Sélectionner les n-uplets de R par l'index sur $R.A$ muni de la clé 'c'
2. Pour chaque valeur de $R.C$, utiliser l'index sur $S.C$ pour retrouver les n-uplets en correspondance
3. Éliminer les n-uplets de S tels que $S.E \neq 2$
4. Concaténer les n-uplets de R et de S
5. Projeter sur B et D puis ajouter au résultat.

...et son illustration

R

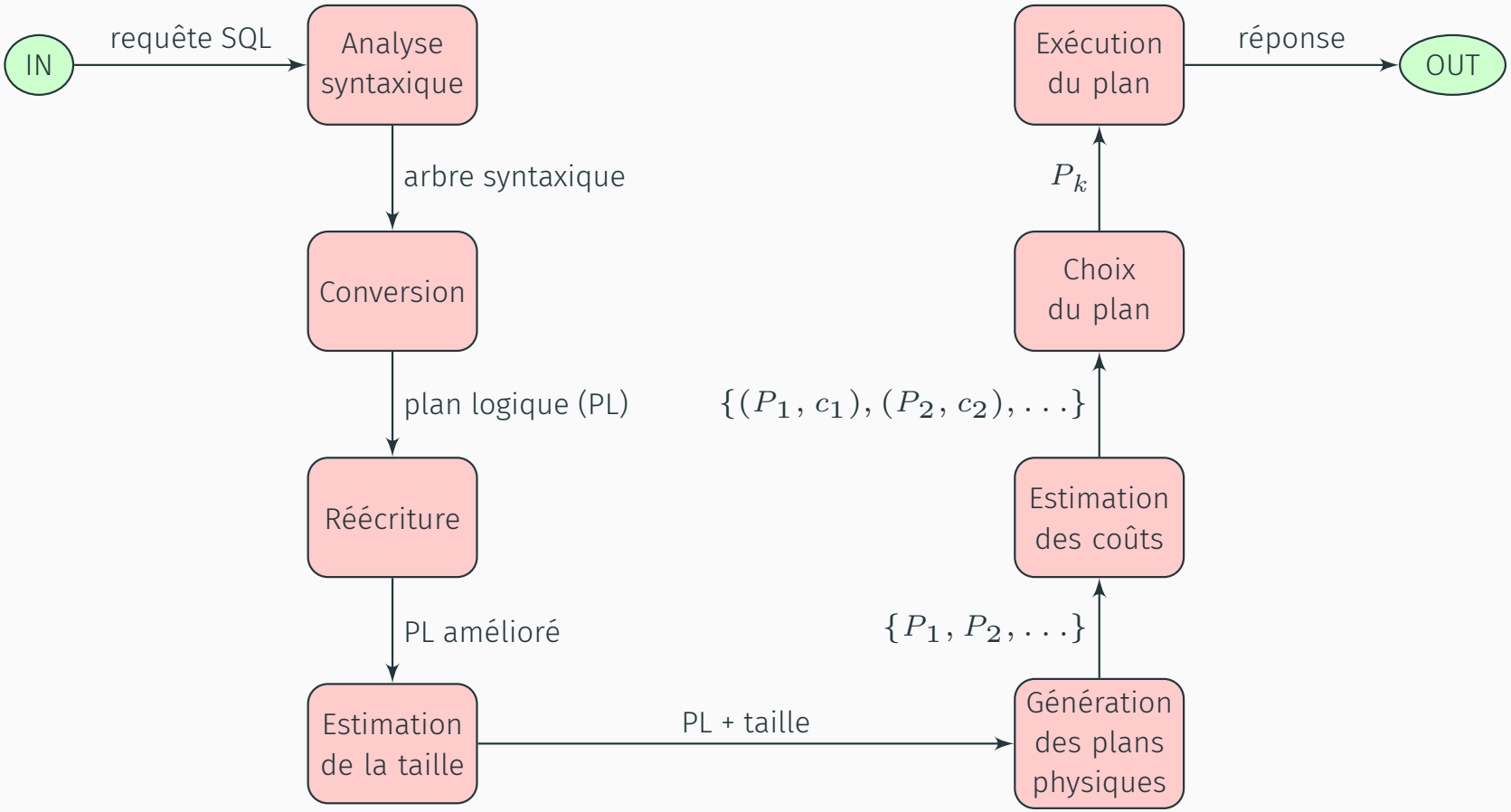
A	B	C
a	1	10
b	1	20
c	2	10
d	2	35
e	3	45

1. index sur $R.A$, clé ‘c‘ : $\langle c, 2, 10\rangle$
2. index sur $S.C$, clé 10 : $\langle 10, x, 2\rangle$
3. filtre par $S.E = 2$: OK
4. assemblage du n-uplet : $\langle c, 2, 10, x, 2\rangle$
5. projection sur B et D : $\langle 2, x\rangle$
6. retour au point 1.

S

C	D	E
10	x	2
20	y	2
30	z	2
40	x	1
50	y	3

Schéma général pour l'évaluation de requête



Les grandes unités

Évaluation : *compilation* + exécution

Compilation : analyse syntaxique + conversion + *optimisation*

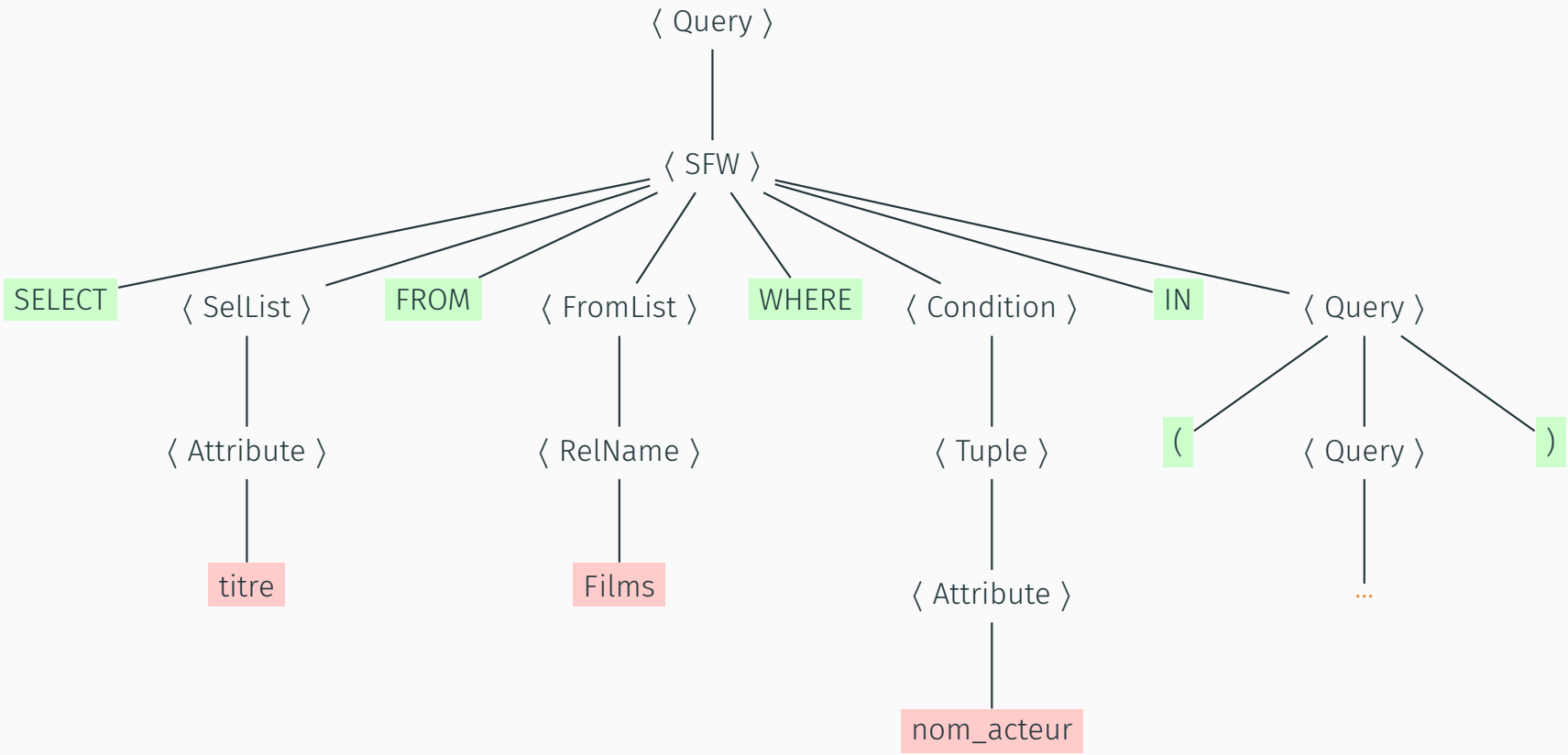
Optimisation : réécriture PL + estimation taille + génération PP + estimation coûts + choix PP

Par l’exemple

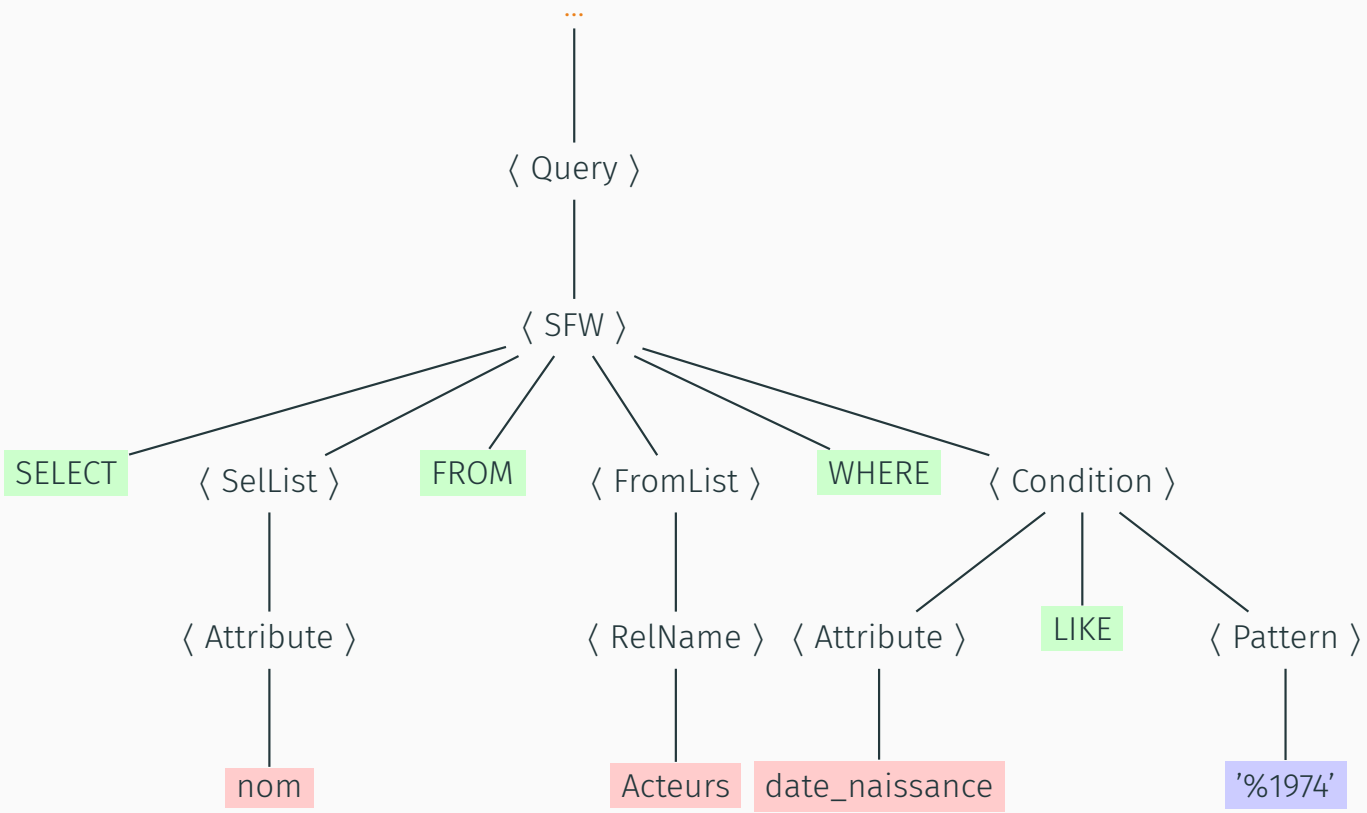
```
SELECT titre
FROM Films
WHERE nom_acteur IN (
    SELECT nom
    FROM Acteurs
    WHERE date_naissance LIKE '%1974'
);
```

(trouver les films avec les acteurs nés en 1974)

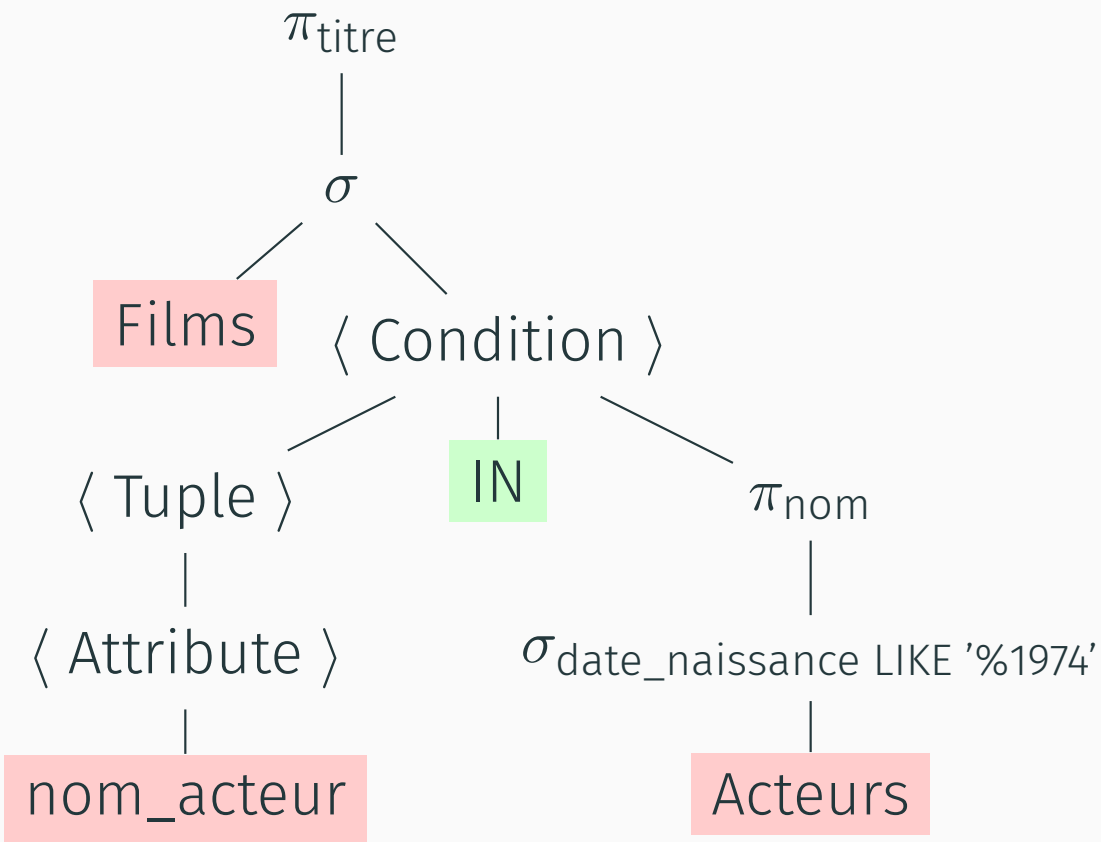
Arbre syntaxique



Arbre syntaxique (suite)

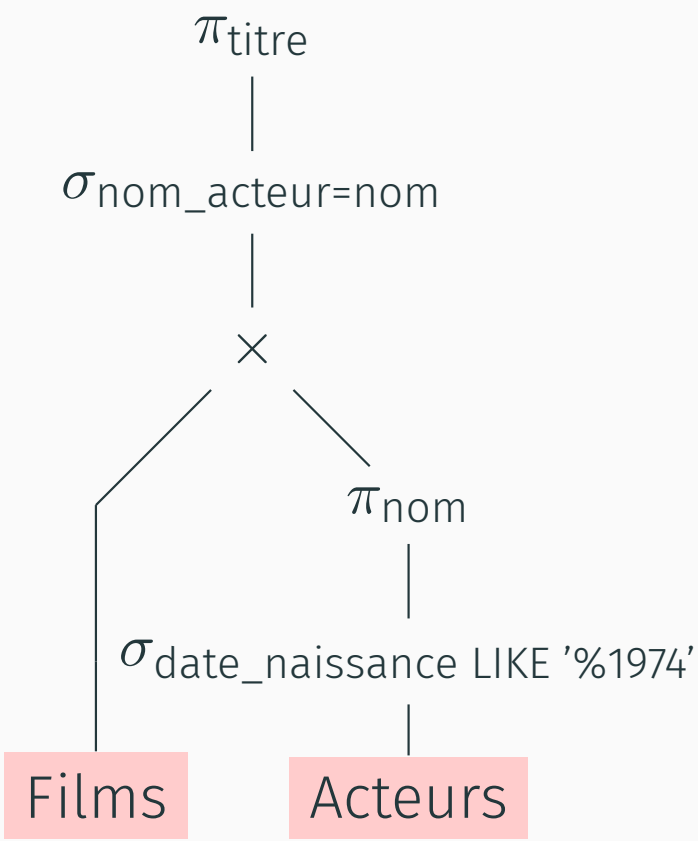


Vers le plan d'exécution



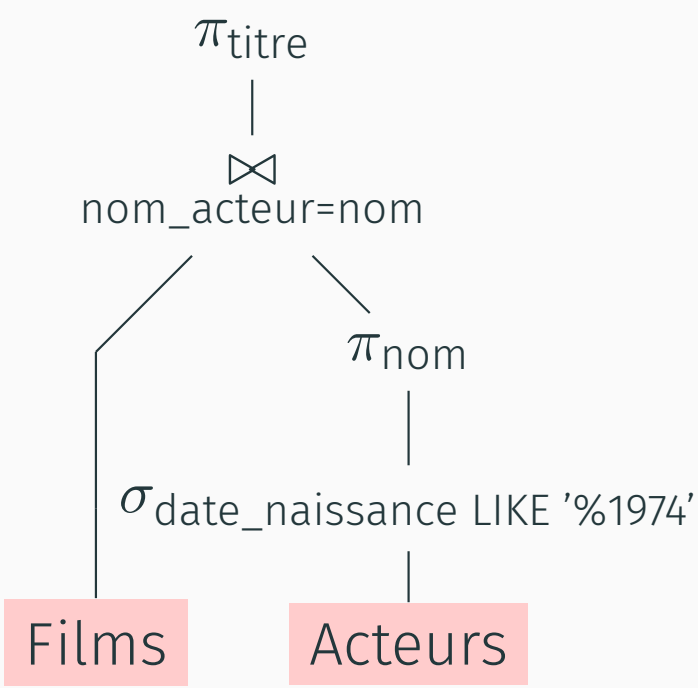
Note : représentation avec une restriction σ binaire, phase transitoire entre l'arbre de syntaxe abstraite et l'expression algébrique.

Plan d'exécution logique



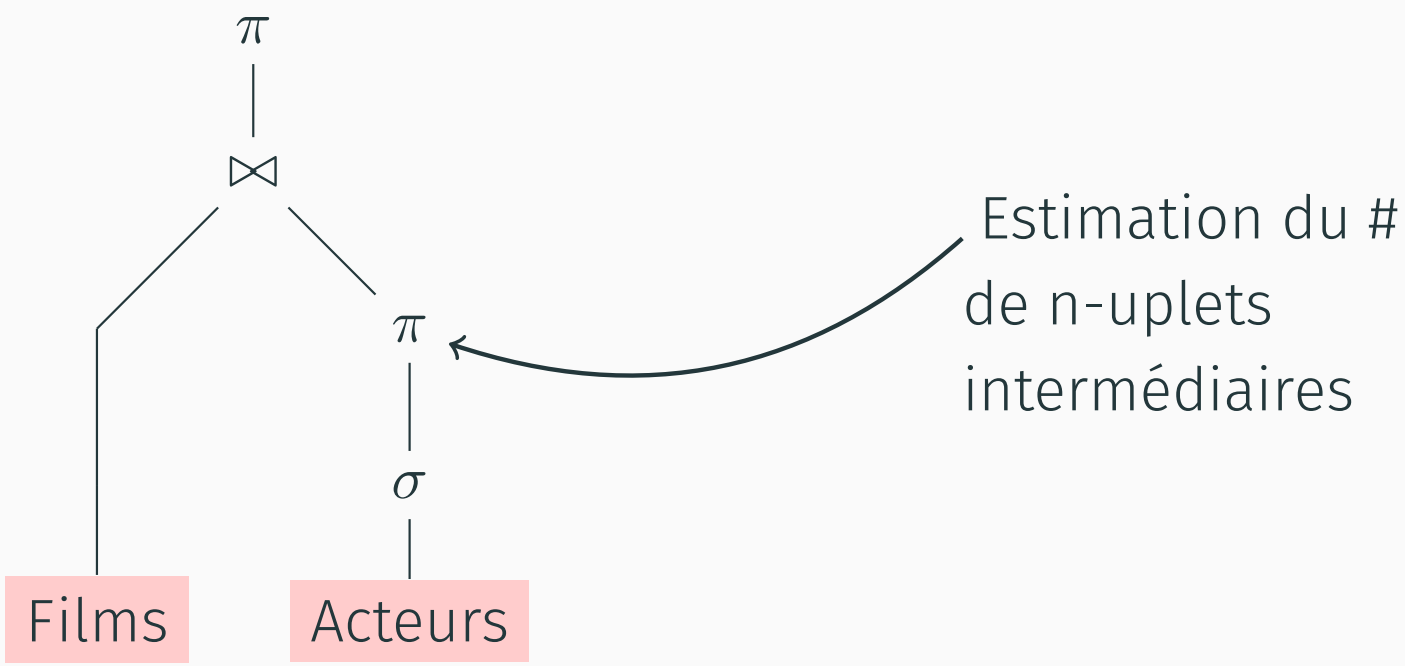
Note :
absorption de la condition IN.

Plan d'exécution logique amélioré

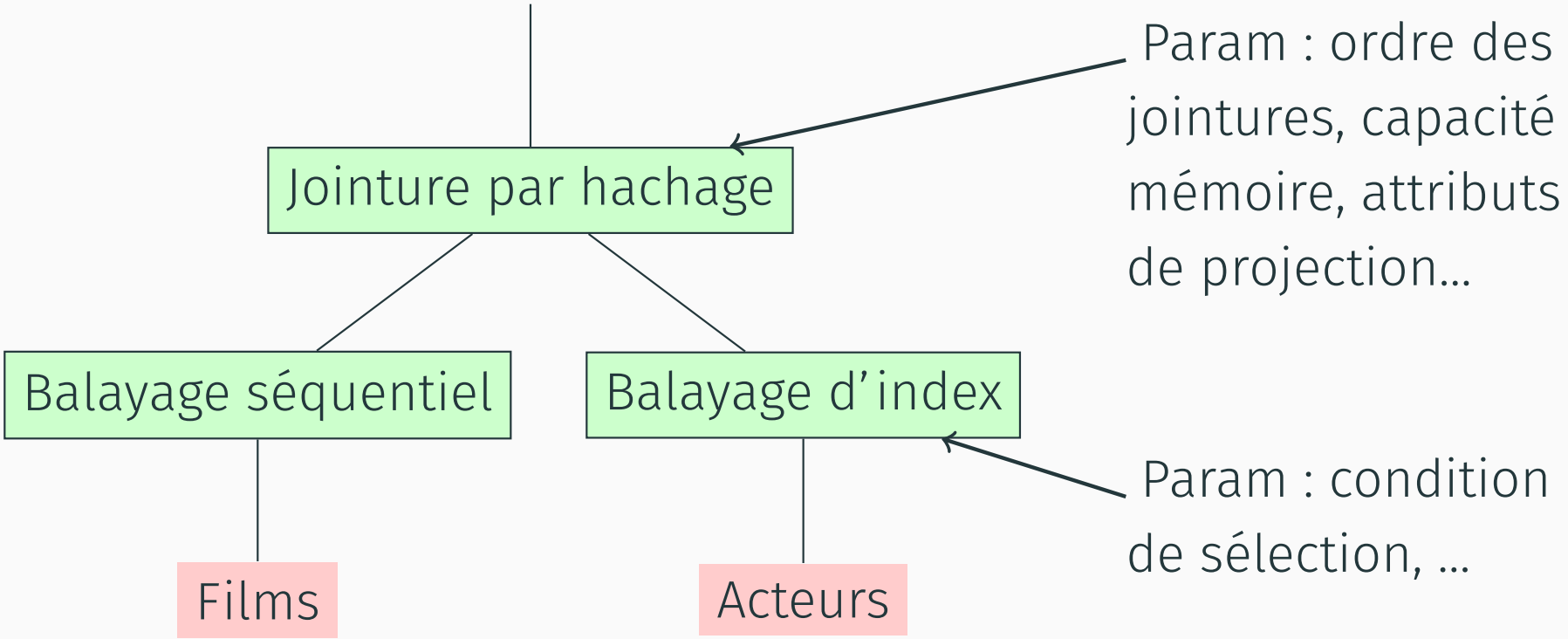


Question : quelles projections *descendre* dans la branche de `Films` ?

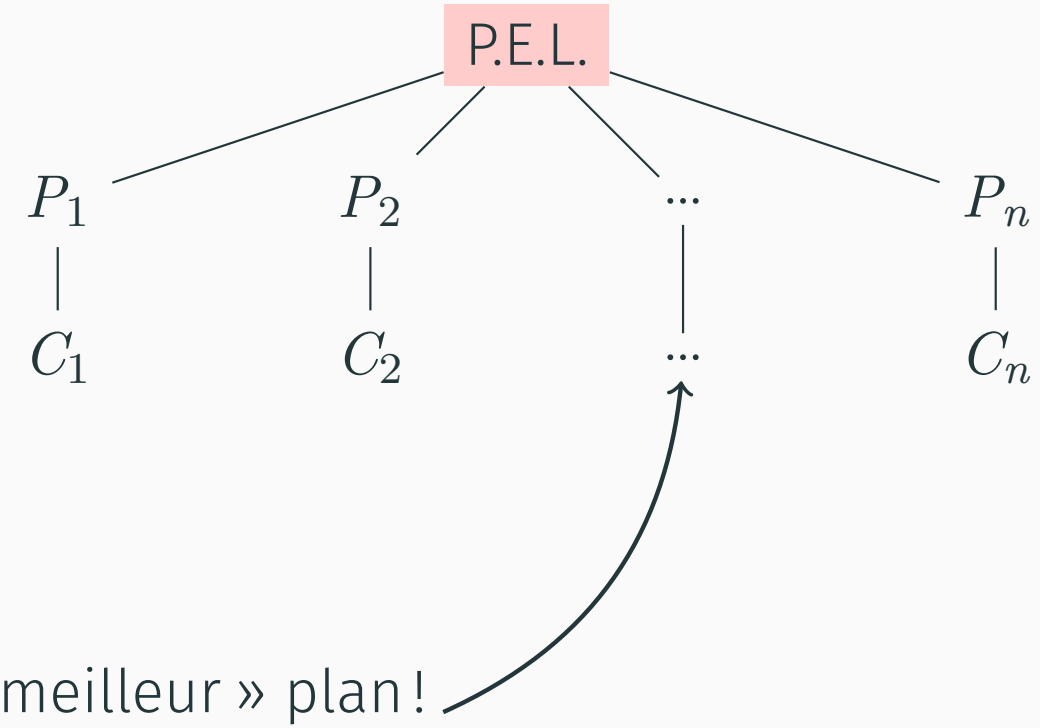
Taille du résultat



Un plan physique



Estimation de coûts



Choisir le « meilleur » plan !

Réécriture de requêtes

Où en est-t-on ?

Aperçu de la chaîne de traitement

Réécriture de requête

Estimation de taille

En route vers le plan physique

Estimation du coût I/O

Éléments de tuning pour la performance

Réécriture de la requête SQL

- Simplification par des expressions équivalentes
- Activité difficile, très ouverte, et potentiellement lucrative

Équivalences logiques

Example

```
SELECT * FROM R
WHERE ( (R.A=a) OR (R.B=1) OR (R.B=2) )
AND NOT ( (R.B=1) OR (R.B=2) )
```

Propriété logique : $((p \vee q \vee r) \wedge \neg(q \vee r)) \Leftrightarrow (p \wedge \neg(q \vee r))$

Expression simplifiée de la requête

```
SELECT * FROM R
WHERE (R.A=a)
AND NOT ( (R.B=1) OR (R.B=2) )
```

Jeu avec les contraintes d'intégrité

Example (X est une clé, i.e. $X \to XYZ$)

$$Q = \pi_{XY}(R) \bowtie \pi_{XZ}(R)$$

La requête se simplifie en $Q = R$

Example ($R.A \in \{a, b, c\} \Rightarrow R.B \geq 3$)

```
SELECT * FROM R
WHERE R.A=b AND R.B <= 2
```

Réponse vide, quelle que soit l'instance de la base de données

Enrichissement de prédicat

Example (Pour autoriser $(R \bowtie T) \bowtie S$)

avant...

```
SELECT * FROM R, S, T
WHERE R.a = S.b AND S.b = T.c
```

après

```
SELECT * FROM R, S, T
WHERE R.a = S.b AND S.b = T.c
      AND R.a = T.c
```

Dans la version originale, $R \bowtie T$ est un produit cartésien

Bloc d'optimisation

Chaque instruction **SELECT** est un *bloc d'optimisation* séparé
$\rightsquigarrow$ optimisation locale

Une sous-requête (ou requête imbriquée) est optimisée *séparément* de la requête principale

Requête imbriquée

Elle apparaît dans toutes les clauses : `SELECT`, `FROM` et `WHERE`

Cas particulier de la clause `WHERE`

- produit une valeur : `X = ( SELECT A FROM... )`
- produit plusieurs valeurs : `X IN ( SELECT A FROM... )`
- mobilise les prédicats `IN ALL ANY EXISTS`, éventuellement négatifs (`NOT`)

Les requêtes simples

Aucune dépendance entre la requête principale et la *requête imbriquée simple*

Example (Requête simple)

```
SELECT O.Qté FROM Commande O
WHERE O.Prix = ( SELECT P.Prix FROM Commande P
                 WHERE P.CommandeId = 7 )
```

Une requête imbriquée simple n'est évaluée qu'*une seule fois*

Les requêtes corrélées

Example (Requête corrélée)

```
SELECT * FROM Commande O
WHERE O.ClientId IN ( SELECT C.ClientId
FROM Client C WHERE C.Solde < O.Prix )
```

Elles sont évaluées *pour chaque nouvelle valeur* de la requête principale

Optimisation

Élicitation de la jointure $\texttt{COMMANDE} \bowtie \texttt{CLIENT}$

Réécriture de l'expression algébrique

- Règles de transformation : expressions équivalentes
- Quelles sont les bonnes transformations ?

Les règles

Jointure naturelle...

$$R \bowtie S = S \bowtie R$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

À propos de l'ordre des jointures

Note

- Les noms d'attributs font partie du résultat, donc le sens de la jointure ($R \bowtie S$ ou $S \bowtie R$) n'est pas significatif
- Représentation arborescente de l'ordre des jointures :

Les règles

Jointure naturelle, produit cartésien, union

$$R \bowtie S = S \bowtie R$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$R \times S = S \times R$$

$$(R \times S) \times T = R \times (S \times T)$$

$$R \cup S = S \cup R$$

$$R \cup (S \cup T) = (R \cup S) \cup T$$

Les règles

Restriction (a.k.a. Sélection)

$$\sigma_{p_1,p_2}(R) = \sigma_{p_1}[\sigma_{p_2}(R)]$$
$$\sigma_{p_1 \text{ OU } p_2}(R) = [\sigma_{p_1}(R)] \cup [\sigma_{p_2}(R)]$$

Ensembles & multi-ensembles

$$R = \{a, a, b, b, b, c\}$$
$$S = \{b, b, c, c, d\}$$
$$R \cup S = ?$$

- Option 1 : Somme

$$R \cup S = \{a, a, b, b, b, b, b, c, c, c, d\}$$

- Option 2 : Maximum

$$R \cup S = \{a, a, b, b, b, c, c, d\}$$

Ce qui fonctionne bien

L'option 2 (MAX) valide la règle suivante :

$$\sigma_{p_1 \text{ OU } p_2}(R) = \sigma_{p_1}(R) \cup \sigma_{p_2}(R)$$

Example ($R = \{a, a, b, b, b, c\}$)

p_1 vérifié par a, b; p_2 vérifié par b, c

$$\sigma_{p_1 \text{ OU } p_2}(R) = R$$
$$\sigma_{p_1}(R) = \{a, a, b, b, b\} \quad \text{et} \quad \sigma_{p_2}(R) = \{b, b, b, c\}$$
$$\sigma_{p_1}(R) \cup_{\text{MAX}} \sigma_{p_2}(R) = \{a, a, b, b, b, c\}$$

- Résultat pour $\cup_\Sigma$?

Et ce qui est attendu

L'option 1 (Σ) fait sens :

$R =$

A	B
1	a
1	b

$S =$

A	B
1	b
2	b

- Résultat de l'union $R \cup_{\Sigma} S$? $R \cup_{\text{MAX}} S$?
- Résultat de COUNT($R \cup_{\Sigma} S$) ? COUNT($R \cup_{\text{MAX}} S$) ?

Alors : Σ ou MAX ?

- C'est l'option Σ qui vaut pour les unions de multi-ensembles
- Certaines règles ne s'appliquent plus !

Les règles

Projection

Soient :

- X un ensemble d'attributs
- Y un ensemble d'attributs
- $XY = X \cup Y$

$$\pi_X[\pi_Y(R)] = \pi_{XY}(R) \qquad \text{FAUX !}$$

Cascade de projections :

$$\pi_X[\pi_{XY}(R)] = \pi_X(R)$$

Encore des règles

Combinaison $\sigma+\bowtie$

Soient :

- p un prédicat portant sur les attributs de R
- q un prédicat portant sur les attributs de S
- m un prédicat portant sur les attributs de R et de S

$$\sigma_p(R \bowtie S) = [\sigma_p(R)] \bowtie S$$
$$\sigma_q(R \bowtie S) = R \bowtie [\sigma_q(S)]$$

Encore des règles

Combinaison $\sigma+ \bowtie$ (suite)

Règles dérivées :

$$\sigma_{p,q}(R \bowtie S) = [\sigma_p(R)] \bowtie [\sigma_q(S)]$$

$$\sigma_{p,q,m}(R \bowtie S) = \sigma_m[\sigma_p(R) \bowtie \sigma_q(S)]$$

$$\sigma_{p \text{ OU } q}(R \bowtie S) = [\sigma_p(R) \bowtie S] \cup [R \bowtie \sigma_q(S)]$$

Encore des règles

Dérivation.

$$\begin{aligned}\sigma_{p,q}(R \bowtie S) &= \sigma_p[\sigma_q(R \bowtie S)] \\ &= \sigma_p[R \bowtie \sigma_q(S)] \\ &= [\sigma_p(R)] \bowtie [\sigma_q(S)]\end{aligned}$$

□

Les autres dérivations : à faire

Encore plus de règles !

Combinaison $\pi + \sigma$

Soient :

- X un sous-ensemble d'attributs de R
- Z les attributs en jeu dans le prédicat p, sous-ensemble d'attributs de R

$$\pi_X[\sigma_p(R)] = \pi_X[\sigma_p(\pi_{XZ}(R))]$$

Encore plus de règles !

Combinaison $\pi + \bowtie$

Soient :

- X un sous-ensemble d'attributs de R
- Y un sous-ensemble d'attributs de S
- Z l'intersection des attributs de R et de S

$$\pi_{XY}(R \bowtie S) = \pi_{XY}[\pi_{XZ}(R) \bowtie \pi_{YZ}(S)]$$

Toujours des règles...

Combinaison $\pi + \sigma + \bowtie$

$$\pi_{XY}[\sigma_p(R \bowtie S)] = \pi_{XY}[\sigma_p(\pi_{XZ'}(R) \bowtie \pi_{YZ'}(S))]$$

avec $Z' = Z \cup \{\text{attributs utilisés dans } p\}$

Combinaison $\pi + \sigma + \times$

même idée que la jointure...

Par exemple, $\sigma_p(R \times S) = ?$

Les dernières règles!

Combinaisons $\sigma + \cup$ et $\sigma + \setminus$

$$\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$$
$$\sigma_p(R \setminus S) = \sigma_p(R) \setminus S = \sigma_p(R) \setminus \sigma_p(S)$$

La der des ders

Lois pour δ

Élimination de doublons

$$\delta(R \times S) = \delta(R) \times \delta(S)$$

$$\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$$

$$\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$$

$$\delta(\sigma(R)) = \sigma(\delta(R))$$

Opérateurs ensemblistes ($\cup \cap \setminus$) et projection (π) ?

Un peu de ménage

Quelles sont les « bonnes » transformations ?

1. $\sigma_{\{p_1,p_2\}}(R) \to \sigma_{p_1}[\sigma_{p_2}(R)]$
2. $\sigma_p(R \bowtie S) \to [\sigma_p(R)] \bowtie S$
3. $R \bowtie S \to S \bowtie R$
4. $\pi_X[\sigma_p(R)] \to \pi_X[\sigma_p(\pi_{XZ}(R))]$

Et du bon sens

Conventionnellement, on pousse les projections au plus près des relations

Example

Soient

- $R(A, B, C, D, E)$
- $X = \{E\}$
- $p : \{A = a, B = b\}$

$$\pi_X(\sigma_p(R)) \rightarrow \pi_X[\sigma_p(\pi_{XAB}(R))]$$

Attention aux hypothèses

Que se passe-t-il en présence d'indexes sur A et sur B ?

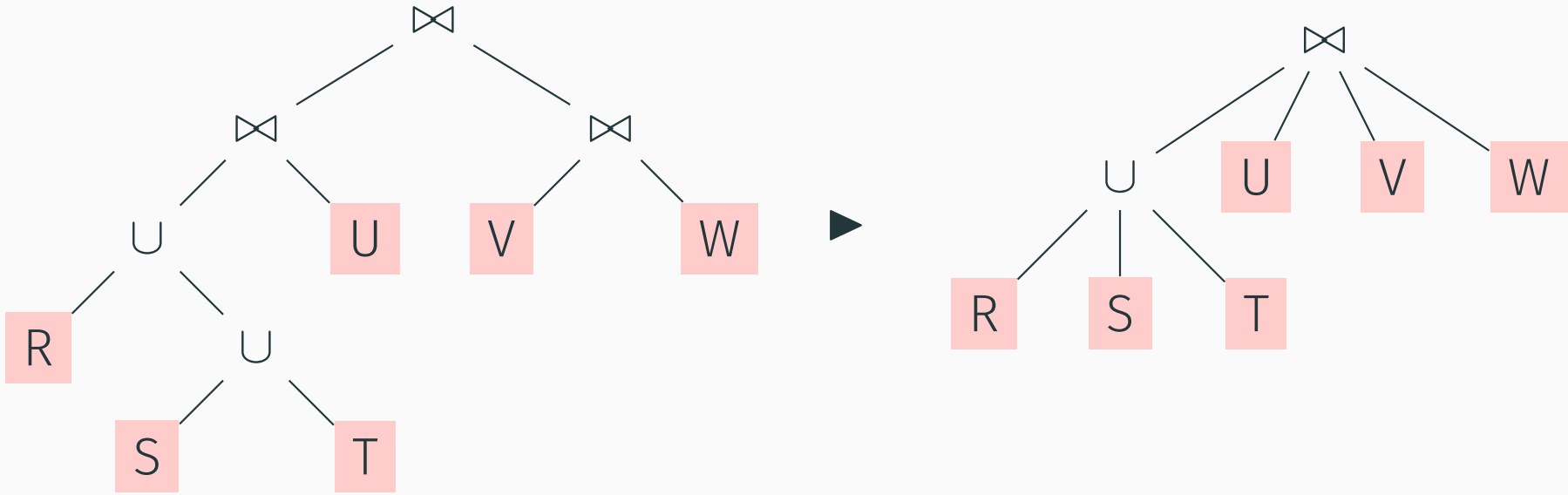
1. L_1 : liste des @ de n-uplets vérifiant $B = b$
2. L_2 : liste des @ de n-uplets vérifiant $A = a$
3. $L_1 \cap L_2$: liste des @ de n-uplets vérifiant p

Principes clé

1. Aucune transformation n'est toujours bonne
2. Suppression des produits cartésiens+restrictions au profit de jointures
3. Habituellement intéressant : *pousser* les restrictions et les projections
    - Réducteurs souvent radicaux du volume de données

En finir avec la réécriture

Opérateurs associatifs-commutatifs



Estimation de taille

Rappel du plan

Aperçu de la chaîne de traitement

Réécriture de requête

Estimation de taille

En route vers le plan physique

Estimation du coût I/O

Éléments de tuning pour la performance

Coût d'un plan de requête

1. Estimer la taille du résultat
2. Estimer # I/O disque

Pour estimer la taille

Conserver des statistiques sur la relation R

- T(R) : # n-uplets dans R
- S(R) : # octets pour chaque n-uplet de R
- B(R) : # pages pour stocker tous les n-uplets de R
- V(R, A) : # valeurs distinctes de l'attribut A dans R

Et en vrai

Example

$R =$

A	B	C	D
chat	1	10	a
chat	1	20	b
chien	1	30	a
chien	1	40	c
rat	1	50	d

- A : 20 octets (chaîne)
- B : 4 octets (entier)
- C : 8 octets (date)
- D : 5 octets (chaîne)

$$\mathsf{T}(R) = 5 \quad \mathsf{S}(R) = 37$$
$$\mathsf{V}(R, A) = 3 \quad \mathsf{V}(R, C) = 5$$
$$\mathsf{V}(R, B) = 1 \quad \mathsf{V}(R, D) = 4$$

Taille du produit cartésien

Estimation pour $W = R_1 \times R_2$

$$\mathsf{T}(W) = \mathsf{T}(R_1) \cdot \mathsf{T}(R_2)$$

$$\mathsf{S}(W) = \mathsf{S}(R_1) + \mathsf{S}(R_2)$$

Taille de la restriction

Estimation pour $W = \sigma_{X=v}(R)$

$$\mathsf{S}(W) = \mathsf{S}(R)$$

$$\mathsf{T}(W) = ?$$

n-uplets de la restriction

$R =$

A	B	C	D
chat	1	10	a
chat	1	20	b
chien	1	30	a
chien	1	40	c
rat	1	50	d

$$\mathsf{V}(R, A) = 3$$
$$\mathsf{V}(R, B) = 1$$
$$\mathsf{V}(R, C) = 5$$
$$\mathsf{V}(R, D) = 4$$

$$W = \sigma_{X=v}(R) \qquad \mathsf{T}(W) = \frac{\mathsf{T}(R)}{\mathsf{V}(R, X)}$$

Répartition sur $R.X$

Hypothèse

Les valeurs en jeu dans le prédicat de sélection $X = v$ sont uniformément réparties sur le domaine actif $\mathrm{adom}(R, X)$ de X

Rappel

$$\mathrm{V}(R, X) = |\,\mathrm{adom}(R, X)|$$

Répartition sur le domaine de $R.X$

Hypothèse alternative

Les valeurs en jeu dans le prédicat de sélection $X = v$ sont uniformément réparties sur le domaine $\mathrm{dom}(R, X)$ de X

Par la pratique

Example

$R =$

A	B	C	D
chat	1	10	a
chat	1	20	b
chien	1	30	a
chien	1	40	c
rat	1	50	d

Hypothèse alternative

$$\mathsf{V}(R, A) = 3 \quad |\operatorname{dom}(R, A)| = 10$$
$$\mathsf{V}(R, B) = 1 \quad |\operatorname{dom}(R, B)| = 10$$
$$\mathsf{V}(R, C) = 5 \quad |\operatorname{dom}(R, C)| = 10$$
$$\mathsf{V}(R, D) = 4 \quad |\operatorname{dom}(R, D)| = 10$$

$$W = \sigma_{X=v}(R) \qquad \mathsf{T}(W) = ?$$

Un détour par la sélectivité

Définition

Estimation du degré de filtrage d'un prédicat, calculée comme la probabilité pour un n-uplet d'être un élément du résultat

- Le résultat attendu est vide si la sélectivité est nulle
- Le résultat attendu est $\Pi_i R_i$ si la sélectivité vaut 1
- Par extension : $\mathrm{sel}(R, A)$ = % d'enregistrements qui vérifient le prédicat d'égalité sur $R.A$

$$\mathrm{sel}(R, A) = \begin{cases} \frac{1}{\mathbb{V}(R,A)} & \\ & \\ \frac{1}{|\mathrm{dom}(R,A)|} & \end{cases}$$

Comptes d'apothicaire

Calcul pour A, B, C sous l'hypothèse alternative

$$
\begin{aligned}
C = v: \quad \mathsf{T}(W) &= \frac{1}{10} \cdot 1 + \frac{1}{10} \cdot 1 + \frac{1}{10} \cdot 1 + \frac{1}{10} \cdot 1 + \frac{1}{10} \cdot 1 \\
&= \frac{5}{10} = 0.5 \\
B = v: \quad \mathsf{T}(W) &= \frac{1}{10} \cdot 5 = 0.5 \\
A = v: \quad \mathsf{T}(W) &= \frac{1}{10} \cdot 2 + \frac{1}{10} \cdot 2 + \frac{1}{10} \cdot 1 \\
&= 0.5
\end{aligned}
$$

Retour à R

$R =$

A	B	C	D
chat	1	10	a
chat	1	20	b
chien	1	30	a
chien	1	40	c
rat	1	50	d

Hypothèse alternative

$$\mathsf{V}(R, A) = 3 \quad |\operatorname{dom}(R, A)| = 10$$
$$\mathsf{V}(R, B) = 1 \quad |\operatorname{dom}(R, B)| = 10$$
$$\mathsf{V}(R, C) = 5 \quad |\operatorname{dom}(R, C)| = 10$$
$$\mathsf{V}(R, D) = 4 \quad |\operatorname{dom}(R, D)| = 10$$

$$W = \sigma_{X=v}(R) \qquad \mathsf{T}(W) = \frac{\mathsf{T}(R)}{|\operatorname{dom}(R, X)|}$$

Prédicat délicat

Comment estimer $W = \sigma_{X \geq v}(R)$?

$$\mathsf{T}(W) = ?$$

- Solution n°1 : $\mathsf{T}(W) = \mathsf{T}(R)/2$
- Solution n°2 : $\mathsf{T}(W) = \mathsf{T}(R)/3$

Encore autrement

Solution n°3

Estimer les valeurs en proportion de l'intervalle

$$R = \begin{array}{|l|l|} \hline Y & X \\ \hline & 1 \to \min \\ \hline & 2 \\ \hline & \ldots \\ \hline & 20 \to \max \\ \hline \end{array}$$

$$\mathsf{V}(R, X) = 10$$

$$W = \sigma_{X \geq 15}(R)$$

$$\mathfrak{f} = \frac{20 - 15 + 1}{20 - 1 + 1} = \frac{6}{20}$$ (fraction de l'intervalle de valeurs)

$$\mathsf{T}(W) = \mathfrak{f} \cdot \mathsf{T}(R)$$

Ou bien

De manière équivalente

$$\mathfrak{f} \cdot \mathsf{V}(R, X) = \text{fraction des valeurs distinctes}$$

$$\mathsf{T}(W) = [\mathfrak{f} \cdot \mathsf{V}(R, X)] \cdot \frac{\mathsf{T}(R)}{\mathsf{V}(R, X)} = \mathfrak{f} \cdot \mathsf{T}(R)$$

Taille de la jointure

Estimation pour $W = R_1 \bowtie R_2$

Soient

- X : attributs de R_1
- Y : attributs de R_2

Cas n°1 $X \cap Y = \emptyset$

Même traitement que $R_1 \times R_2$

Mais encore

Cas n°2 $W = R_1 \bowtie R_2$ **et** $X \cap Y = A$

$R_1 =$

A	B	C

$R_2 =$

A	D

Hypothèses exclusives :

$\mathrm{V}(R_1, A) \leq \mathrm{V}(R_2, A)$: toute valeur de A dans R_1 est dans R_2

$\mathrm{V}(R_2, A) \leq \mathrm{V}(R_1, A)$: toute valeur de A dans R_2 est dans R_1

Hypothèse n°1

Estimation pour la jointure

On suppose l'inclusion des ensembles de valeurs

L'hypothèse est avérée lorsqu'il s'agit d'une équi-jointure selon une contrainte d'intégrité référentielle (clé primaire/clé étrangère)

de n-uplets de la jointure

Calcul de $\mathsf{T}(W)$ **avec** $\mathsf{V}(R_1, A) \leq \mathsf{V}(R_2, A)$

$R_1 =$

A	B	C
7	a	20

$R_2 =$

A	D
4	foo
7	foo
7	bar

1 n-uplet de R_1 s'apparie à $\frac{\mathsf{T}(R_2)}{\mathsf{V}(R_2,A)}$ n-uplets de R_2

$$\text{Donc} \quad \mathsf{T}(W) = \frac{\mathsf{T}(R_2)}{\mathsf{V}(R_2, A)} \cdot \mathsf{T}(R_1)$$

Les deux sens

$$\text{si } \mathsf{V}(R_1, A) \leq \mathsf{V}(R_2, A) \quad \text{alors } \mathsf{T}(W) = \frac{\mathsf{T}(R_2) \cdot \mathsf{T}(R_1)}{\mathsf{V}(R_2, A)}$$

$$\text{si } \mathsf{V}(R_2, A) \leq \mathsf{V}(R_1, A) \quad \text{alors } \mathsf{T}(W) = \frac{\mathsf{T}(R_2) \cdot \mathsf{T}(R_1)}{\mathsf{V}(R_1, A)}$$

A est un attribut partagé par R_1 et R_2

En définitive

Cas général pour $W = R_1 \bowtie R_2$

$$\mathsf{T}(W) = \frac{\mathsf{T}(R_1) \cdot \mathsf{T}(R_2)}{\max(\mathsf{V}(R_1, A), \mathsf{V}(R_2, A))}$$

Une autre méthode

Cas n°2 avec des hypothèses alternatives

Les valeurs sont uniformément réparties sur le domaine de $R.A$

$R_1 =$

A	B	C
7	a	20

$R_2 =$

A	D
4	foo
7	foo
7	bar

1 n-uplet de R_1 s'apparie à $\frac{\mathsf{T}(R_2)}{|\mathrm{dom}(R_2,A)|}$ n-uplets de R_2

$$\text{Donc} \quad \mathsf{T}(W) = \frac{\mathsf{T}(R_2) \cdot \mathsf{T}(R_1)}{|\mathrm{dom}(R_2, A)|} = \frac{\mathsf{T}(R_2) \cdot \mathsf{T}(R_1)}{|\mathrm{dom}(R_1, A)|}$$

Volume d'un n-uplet

Dans tous les cas

$$\mathrm{S}(W) = \mathrm{S}(R_1) + \mathrm{S}(R_2) - \mathrm{S}(A)$$

$\mathrm{S}(A)$ taille de l'attribut A

Pot pourri

Avec des principes similaires

$\pi_{AB}(R) \ldots$

$\sigma_{\{A=a,B=b\}}(R) \ldots$

$R_1 \bowtie R_2$ avec les attributs communs A, B, C

$\cup \quad \cap \quad \setminus \ldots$

Au sujet des expressions complexes

Calcul de résultats intermédiaires pour T, S, V

Example

$$W = [\quad \underbrace{\sigma_{A=a}(R_1)}_{\text{en tant que relation } U} \quad] \bowtie R_2$$

$$\mathsf{T}(U) = \mathsf{T}(R_1)/\,\mathsf{V}(R_1, A) \qquad \mathsf{S}(U) = \mathsf{S}(R_1)$$

Qu'en est-il de $\mathsf{V}(U, *)$?

Estimer les V's

$U = \sigma_{A=a}(R_1)$

Soit $R_1(A, B, C, D)$

$$\mathsf{V}(U, A) =$$

$$\mathsf{V}(U, B) =$$

$$\mathsf{V}(U, C) =$$

$$\mathsf{V}(U, D) =$$

On regarde ce que ça donne

$R_1 =$

A	B	C	D
chat	1	10	a
chat	1	20	b
chien	1	30	a
chien	1	40	c
rat	1	50	a

$$\mathsf{V}(R_1, A) = 3$$
$$\mathsf{V}(R_1, B) = 1$$
$$\mathsf{V}(R_1, C) = 5$$
$$\mathsf{V}(R_1, D) = 3$$

$$U = \sigma_{A=a}(R_1)$$

$$\mathsf{V}(U, A) = 1 \quad \mathsf{V}(U, B) = 1 \quad \mathsf{V}(U, C) = \frac{\mathsf{T}(R_1)}{\mathsf{V}(R_1, A)}$$

$\mathsf{V}(U, D)$ ...quelque part entre $\mathsf{V}(U, B)$ et $\mathsf{V}(U, C)$

Une option

Estimation raisonnable pour $U = \sigma_{A=a}(R)$

$$\mathsf{V}(U, A) = 1$$
$$\mathsf{V}(U, X) = \mathsf{V}(R, X)$$

Le complément pour la jointure

$$U = R_1(A, B) \bowtie R_2(A, C)$$

$$\mathsf{V}(U, A) = \min(\mathsf{V}(R_1, A), \mathsf{V}(R_2, A))$$

$$\mathsf{V}(U, B) = \mathsf{V}(R_1, B) \qquad \mathsf{V}(U, C) = \mathsf{V}(R_2, C)$$

Hypothèse n°2

Estimation des V's secondaires

On préserve l'ensemble des valeurs

Les 2 papillons

Example

$$Z = R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D)$$

$\boxed{R_1}$	$\mathsf{T}(R_1) = 1000$	$\mathsf{V}(R_1, A) = 50$	$\mathsf{V}(R_1, B) = 100$
$\boxed{R_2}$	$\mathsf{T}(R_2) = 2000$	$\mathsf{V}(R_2, B) = 200$	$\mathsf{V}(R_2, C) = 300$
$\boxed{R_3}$	$\mathsf{T}(R_3) = 3000$	$\mathsf{V}(R_3, C) = 90$	$\mathsf{V}(R_3, D) = 500$

D’abord le premier

Résultat partiel $U = R_1 \bowtie R_2$

$$\mathsf{T}(U) = \frac{1000 \times 2000}{200}$$

$$\mathsf{V}(U, A) = 50$$

$$\mathsf{V}(U, B) = 100$$

$$\mathsf{V}(U, C) = 300$$

Puis le second

$$Z = U \bowtie R_3$$

$$\mathsf{T}(Z) = \frac{1000 \times 2000 \times 3000}{200 \times 300}$$

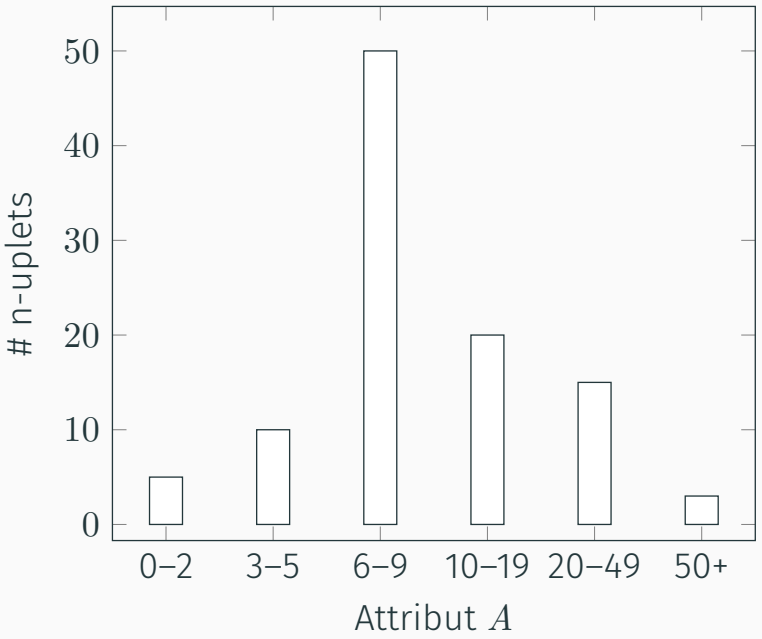
$$\mathsf{V}(Z, A) = 50$$

$$\mathsf{V}(Z, B) = 100$$

$$\mathsf{V}(Z, C) = 90$$

$$\mathsf{V}(Z, D) = 500$$

À propos d'histogrammes



n-uplets de R avec une valeur de A dans un intervalle donné

$$\sigma_{A=a}(R) = ?$$

Synthèse

- Estimer la taille du résultat est un « art »
- Ne pas oublier :
    - Les statistiques doivent être rafraîchies
    - (à quel coût ?)

Plan physique

Où en est-on ?

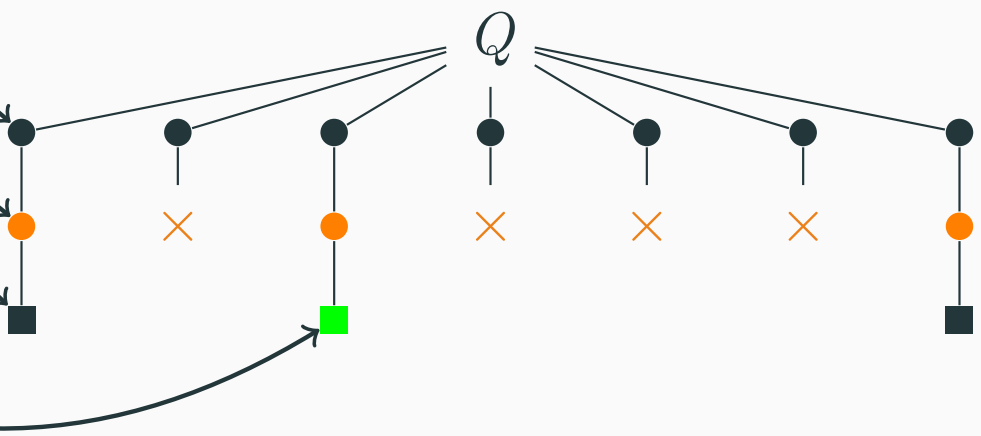
Récapitulatif

- ...
- Construction du plan logique amélioré
- Estimation de coût
  - Taille du résultat ⟵ ok
  - # I/O disque ⟵ à faire
- Génération et comparaison des plans physiques

Principe de l'optimisation

Générer et comparer les plans

1. Générer
2. Élaguer
3. Estimer le coût
4. Sélectionner



Les ingrédients

Pour la génération de plans

- Transformer les expressions algébriques
- Varier l'ordre des jointures
- Fixer les méthodes d'accès
- Construire de nouveaux indexes ou trier à la volée
- Déterminer les détails d'implémentation
  - algorithmes de jointure
  - gestion de la mémoire
  - *etc.*

Ordre d'exécution des jointures

Une question de combinatoire

- $\frac{(2n)!}{n!}$ arbres d'exécution de la jointure de $n+1$ relations
- Sans tenir compte des choix d'implémentation, *etc.*

# relations	arbres
2	2
3	12
4	120
6	30 240
8	17 297 280
10	17 643 225 600

Trop de plans possibles !

- Élagage de l'espace de recherche

Méthode traditionnelle : *programmation dynamique*

- Trouver le meilleur plan pour la jointure de n relations en n itérations
- À chaque itération k, trouver le meilleur plan partiel pour k relations à partir des meilleurs plans partiels i et $k - i$ $(1 \leq i \leq k)$
- Hypothèse : *principe d'optimalité*

Ordonnancement des jointures par l'exemple 1/5

Cas de la jointure $R \bowtie S \bowtie T \bowtie U$ avec $\mathsf{T}(\cdot) = 1\,000$

Les statistiques

$R(a, b)$	$S(b, c)$	$T(c, d)$	$U(d, a)$
$\mathsf{V}(R, a) = 100$			$\mathsf{V}(U, a) = 50$
$\mathsf{V}(R, b) = 200$	$\mathsf{V}(S, b) = 100$		
	$\mathsf{V}(S, c) = 500$	$\mathsf{V}(T, c) = 20$	
		$\mathsf{V}(T, d) = 50$	$\mathsf{V}(U, d) = 1\,000$

Ordonnancement des jointures par l'exemple 2/5

État initial

	$\{R\}$	$\{S\}$	$\{T\}$	$\{U\}$
Taille	$1\,000$	$1\,000$	$1\,000$	$1\,000$
Coût	0	0	0	0
Meilleur plan	R	S	T	U

Ordonnancement des jointures par l'exemple 3/5

Deuxième tour

	$\{R, S\}$	$\{R, T\}$	$\{R, U\}$	$\{S, T\}$	$\{S, U\}$	$\{T, U\}$
Taille	$5\,000$	10^6	$10\,000$	$2\,000$	10^6	$1\,000$
Coût	0	0	0	0	0	0
Meilleur plan	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

Heuristique n°1

Le plus petit des 2 arguments est placé à gauche

Pourquoi ?

Ordonnancement des jointures par l'exemple 4/5

Troisième tour

	$\{R, S, T\}$	$\{R, S, U\}$	$\{R, T, U\}$	$\{S, T, U\}$
Taille	$10\,000$	$50\,000$	$10\,000$	$2\,000$
Coût	$2\,000$	$5\,000$	$1\,000$	$1\,000$
Meilleur plan	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$

Le coût représente le volume de n-uplets *intermédiaires*, à l'exclusion de la taille du résultat

Ordonnancement des jointures par l'exemple 5/5

Quatrième et dernier tour

Ordonnancement	Coût
$((S \bowtie T) \bowtie R) \bowtie U$	$12\,000$
$((R \bowtie S) \bowtie U) \bowtie T$	$55\,000$
$((T \bowtie U) \bowtie R) \bowtie S$	$11\,000$
$((T \bowtie U) \bowtie S) \bowtie R$	$3\,000$
$(T \bowtie U) \bowtie (R \bowtie S)$	$6\,000$
$(R \bowtie T) \bowtie (S \bowtie U)$	$2\,000\,000$
$(S \bowtie T) \bowtie (R \bowtie U)$	$12\,000$

Conclusion

On choisit $((T \bowtie U) \bowtie S) \bowtie R$

À propos de programmation dynamique

Remarques

- Propagation des coûts de plans optimaux à chaque itération
- Complexité
  - Temps : $\mathcal{O}(3^n)$
  - Espace : $\mathcal{O}(2^n)$

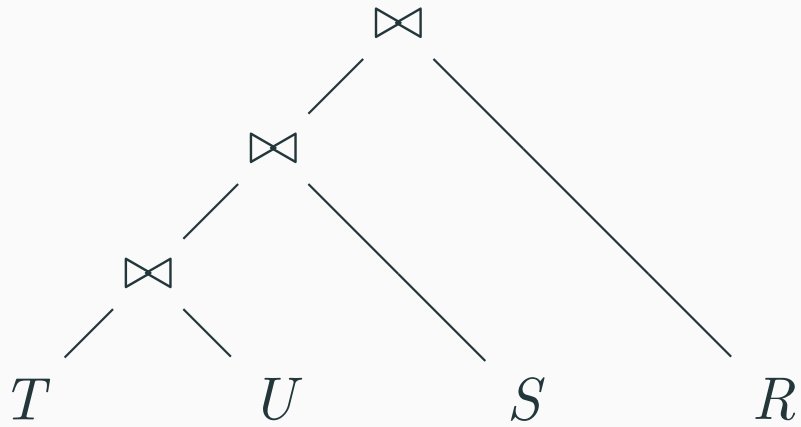
Heuristiques supplémentaires

- Pas de produit cartésien
- Arbres gauche uniquement (vs. arbres « touffus ») : temps réduit à $\mathcal{O}(n \cdot 2^n)$

Plan Linéaire Gauche

Héritage de IBM System R

$((T \bowtie U) \bowtie S) \bowtie R =$



Avantages

- La relation interne (*rhs*) est une table
- Possibilité d'une jointure par index
- Implémentation en *pipeline*

Une algèbre physique

Motivations

Besoin d'un plan d'exécution plus précis que la version algébrique :

- plusieurs méthodes d'accès aux données
- plusieurs algorithmes de calcul des opérateurs algébriques

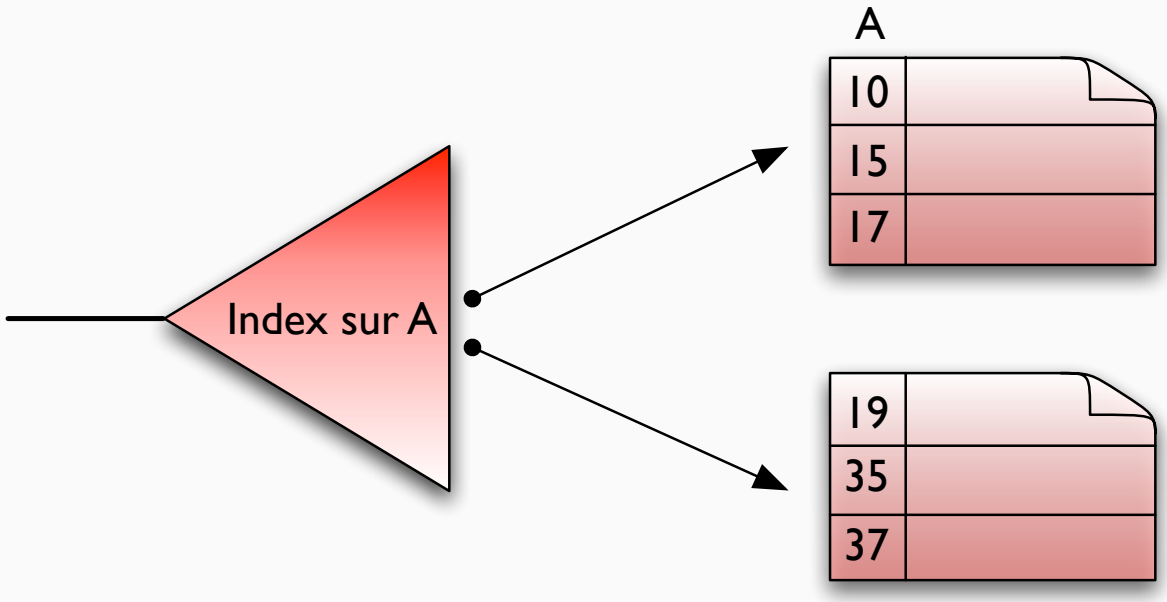
Les méthodes d'accès

Le moyen par lequel les n-uplets sont lus sur disque

- Dépendant de l'organisation physique de la table
- Les options :
  - balayage **séquentiel** : méthode par défaut
  - accès direct, par **index** : récupération des ROWID's selon
    - un prédicat d'égalité $A = v$, ou
    - un intervalle de valeurs $A \in [v_1, v_2]$, ou
    - une combinaison d'indexes sur—un préfixe de—$(A_1, A_2, \ldots A_n)$
  - accès direct, par index plaçant (**cluster**)
  - accès direct, par code de **hachage**

Index plaçant (ou *cluster*)

Index qui permet la lecture des n-uplets selon l'ordre qui correspond à l'organisation physique



Pour mémoire

Notions de groupement (*clustering*)

- Organisation de fichiers groupés

  $\boxed{r_1 r_2 s_1 s_2}$ $\boxed{r_3 r_4 s_3 s_4}$ ...

- Relation groupée

  $\boxed{r_1 r_2 r_3 r_4}$ $\boxed{r_5 r_6 r_7 r_8}$ ...

- Index plaçant (ou groupant)

Les opérateurs physiques

À titre d'exemple :

$\texttt{TableScan}(R)$	$\texttt{Filter}(C)$
$\texttt{RandomScan}(R, X)$	$\texttt{Hash/Merge/NestedLoop Join}(C)$
$X \leftarrow \texttt{IndexScan}(R, C)$	$\texttt{Project}(L)$
$\texttt{SortScan}(R, L)$	$\texttt{Sort}(L)$
$\texttt{HashScan}(R)$	

- X est un ensemble d'adresses (ROWID's)
- L est une liste d'attributs
- C est une condition de type $A = v$ ou $A \in [v_1, v_2]$

Évaluation de requête

La démarche

1. élaboration du plan d'exécution physique
2. accès aux données
3. exécution des algorithmes pour chaque opérateur du plan

Transmission des données

- Objectif : minimiser le nombre I/O de pages
- Approche : évaluation **itérative** en *pipeline*
  - production continue des résultats
  - premiers résultats très rapidement
  - stockage intermédiaire inutile

Pipeline et itérateurs

Tous les opérateurs de l'algèbre physique présentée supportent une architecture en pipeline, sauf le tri

Implémentation des opérateurs sous la forme d'**itérateurs**

- `initialize`() et `next`()
- consommation des entrées une à une
- production des sorties une à une
- propagation des demandes : racine $\rightarrow$ feuilles
- propagation des données : feuilles $\rightarrow$ racine

Choix d'un plan d'exécution

- Obsolète : règles de priorité
  Si $|R| > |S|$ alors P_1 sinon P_2
- Actuel : élaboration d'un *modèle de coût*

Estimation du coût I/O

- Compter le # pages disque lues (et écrites) pour exécuter un plan de requête

Quelques paramètres indispensables

- B(R) : # pages contenant les n-uplets de R
- $\tau(R)$: # maximum de n-uplets de R par page
- $\mathcal{M}$: # pages disponibles en mémoire centrale
- HT(i) : # niveaux (hauteur) de l'index i
- LB(i) : # pages feuilles de l'index i

Estimation de coût

Exemple : $R_1 \bowtie R_2$ sur l'attribut commun C

$$\mathsf{T}(R_1) = 10\,000$$
$$\mathsf{T}(R_2) = 5\,000$$
$$\tau(R_1) = \tau(R_2) = 10$$
$$\mathsf{S}(R_i) = \frac{1}{\tau(R_i)} = 1/10 \text{ page}, i \in \{1, 2\}$$
$$\mathcal{M} = 101 \text{ pages}$$

Métrique : # I/O (on ignore l'écriture du résultat final)

Avertissement

Critère d'estimation incomplet

- on ne comptabilise pas le coût CPU
- on ne comptabilise pas le temps d'exécution
- on ne tient pas compte de l'usage du double tampon (*buffering*), de la lecture en avance *prefetching*, etc.

Concentration sur la jointure naturelle

Les options

- Transformation : $R_1 \bowtie R_2$, $R_2 \bowtie R_1$
- Algorithmes de jointure
  - Itération (boucles imbriquées) — *nested loop join*
  - Fusion — *sort-merge join*
  - Avec index — *index nested loop join*
  - Hachage — *hash join*

Premier algorithme

Principe de la jointure par itération

Pour chaque r dans R_1 faire

pour chaque s dans R_2 faire

si $r.C = s.C$ alors conserver le couple (r, s)

Deuxième algorithme

Principe de la jointure par fusion

1. Si R_1 et R_2 ne sont pas triées, les trier
2. $i \leftarrow 1; j \leftarrow 1;$
   Tant que $(i \leq \mathsf{T}(R_1)) \wedge (j \leq \mathsf{T}(R_2))$ faire
   si $R_1\{i\}.C = R_2\{j\}.C$ alors *conserver_nuplets*
   sinon
   si $R_1\{i\}.C > R_2\{j\}.C$ alors $j \leftarrow j + 1$
   sinon
   si $R_1\{i\}.C < R_2\{j\}.C$ alors $i \leftarrow i + 1$

Suite du 2e algorithme

Procédure conserver_nuplets

Tant que $(R_1\{i\}.C = R_2\{j\}.C) \wedge (i \leq \mathsf{T}(R_1))$ faire

{

$jj \leftarrow j$;

tant que $(R_1\{i\}.C = R_2\{jj\}.C) \wedge (jj \leq \mathsf{T}(R_2))$ faire

conserver le couple $(R_1\{i\}, R_2\{jj\})$; $jj \leftarrow jj + 1$

$i \leftarrow i + 1$

}

Et dans la vraie vie

Exemple

i	$R_1\{i\}.C$	$R_2\{j\}.C$	j
1	10	5	1
2	20	20	2
3	20	20	3
4	30	30	4
5	40	30	5
		50	6
		60	7

Troisième algorithme

Principe de la jointure avec index

Pré-condition : index sur $R_2.C$

Pour chaque r dans R_1 faire

{

$E \leftarrow \text{index}(R_2, C, r.C)$;

pour chaque s dans E faire

conserver le couple (r, s)

}

Remarque à propos de l'instruction $E \leftarrow \text{index}(R, X, v)$:

E contient l'ensemble des n-uplets vérifiant $X = v$

Quatrième algorithme

Principe de la jointure par hachage

Pré-conditions :

- fonction de hachage f prenant ses valeurs entières dans $[0, k]$
- Classes de R_1 : $G_0, G_1, \ldots, G_k$
- Classes de R_2 : $H_0, H_1, \ldots, H_k$

1. Disperser les n-uplets de R_1 dans les G_i en fonction de f
2. Disperser les n-uplets de R_2 dans les H_i en fonction de f
3. Pour $i = 0$ à k faire
   apparier les n-uplets de G_i avec ceux de H_i

Et concrètement

Exemple : Hachage pair/impair

R_1
2
4
3
5
8
9

R_2
5
4
12
3
13
8
11
14

Les classes

	R_1	R_2
Pair :	2 4 8	4 12 8 14
Impair :	3 5 9	5 3 13 11

À quoi faut-il être attentif ?

Facteurs qui affectent les performances

1. Est-ce que les n-uplets des relations sont physiquement rassemblés sur disque ?
2. Est-ce que les relations sont triées sur l'attribut de jointure ?
3. Existe-il des indexes ?

Coût I/O

Exemple 1(a) – Jointure $R_1 \bowtie R_2$ par itération

- Hypothèse Relations non contiguës
- Rappel :

$$\begin{cases} \mathsf{T}(R_1) = 10\,000 \qquad \mathsf{T}(R_2) = 5\,000 \\ \mathsf{S}(R_1) = \mathsf{S}(R_2) = 1/10 \text{ pages} \\ \mathcal{M} = 101 \text{ pages} \end{cases}$$

- Coût :
  Pour chaque n-uplet de R_1, [lire le n-uplet; lire R_2]
- Total $= 10\,000 \cdot [1 + 5\,000] = 50\,010\,000$ I/Os

Encore un effort

Peut-on mieux faire ?

Avec un usage efficace de la mémoire

1. Lire 100 pages de R_1
2. Parcourir tout R_2 (avec 1 page) et joindre
3. Répéter jusqu'à la fin de la lecture de R_1

Et ça vaut combien ?

Calcul de coût

Pour chaque lot de 100 pages de R_1 :

- Lecture du lot de R_1 : 100 I/Os
- Lecture de R_2 : 5 000 I/Os
- Sous-total : 5 100 I/Os

$$\text{Coût total} = \frac{10\,000}{100} \cdot 5\,100 = 510\,000 \text{ I/Os}$$

50 010 000

Est-ce vraiment le meilleur plan ?

Idée : inversion du sens de la jointure $R_2 \bowtie R_1$

$$\begin{aligned}\text{Coût total} &= \frac{5\,000}{100} \cdot (100 + 10\,000) \\ &= 50 \cdot 10\,100 \\ &= 505\,000 \text{ I/Os}\end{aligned}$$

510 000

Exemple 1(b) – Jointure $R_2 \bowtie R_1$ par itération

- Hypothèse Relations contiguës
- Calcul de coût : pour chaque lot de 100 pages de R_2
  - Lecture du lot de R_2 : 100 I/Os
  - Lecture de R_1 : 10 000/10 = 1 000 I/Os
  - Sous-total : 1 100 I/Os

$$\text{Coût total} = \frac{5\,000/10}{100} \cdot 1\,100 = 5\,500 \text{ I/Os}$$

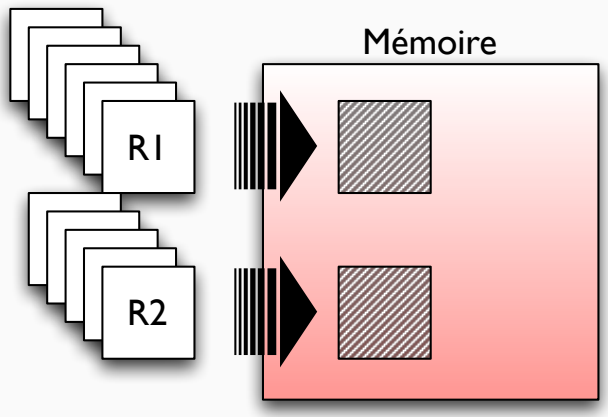
505 000

Et avec un autre algorithme

Exemple 1(c) – Jointure $R_1 \bowtie R_2$ par fusion

- Hypothèses R_1 et R_2 sont triées sur C; les relations sont contiguës

Calcul de coût : lecture de R_1 + lecture de R_2



$$\text{Coût total} = \frac{10\,000}{10} + \frac{5\,000}{10}$$
$$= 1\,500 \text{ I/Os}$$

5.500

Différentes hypothèses

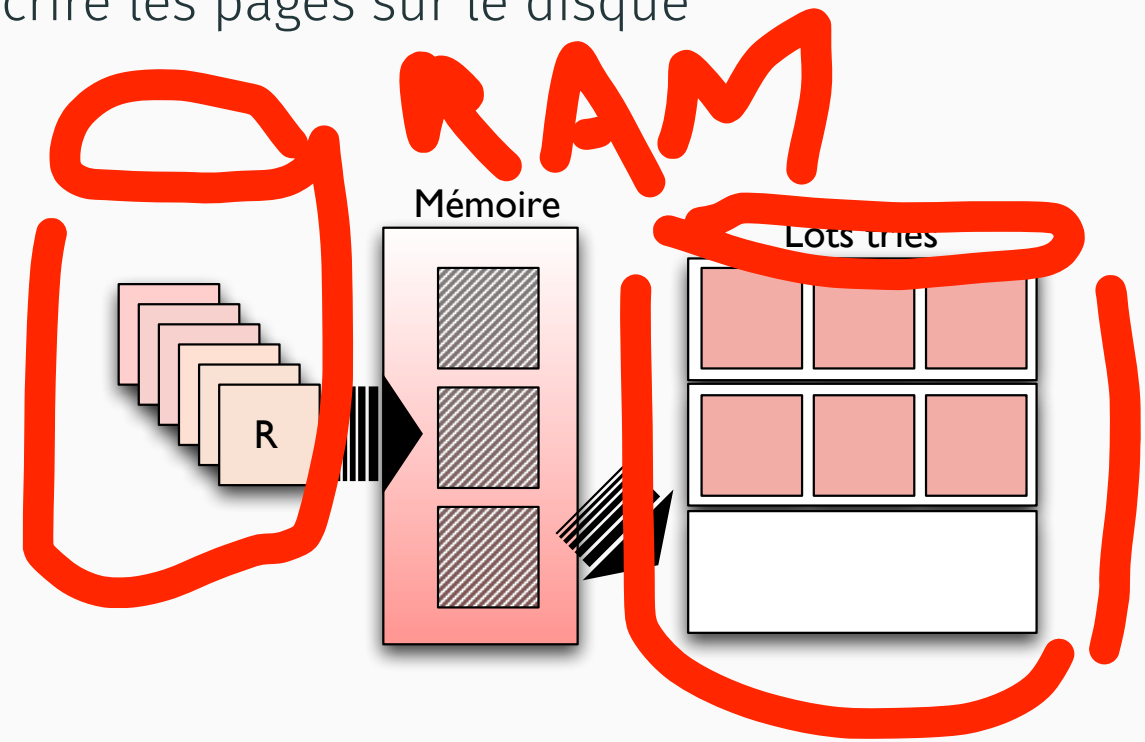
Exemple 1(d) – Jointure $R_1 \bowtie R_2$ par fusion

- Hypothèses R_1 et R_2 non triées mais les relations sont contiguës

Il faut d'abord trier les relations : oui mais comment ?

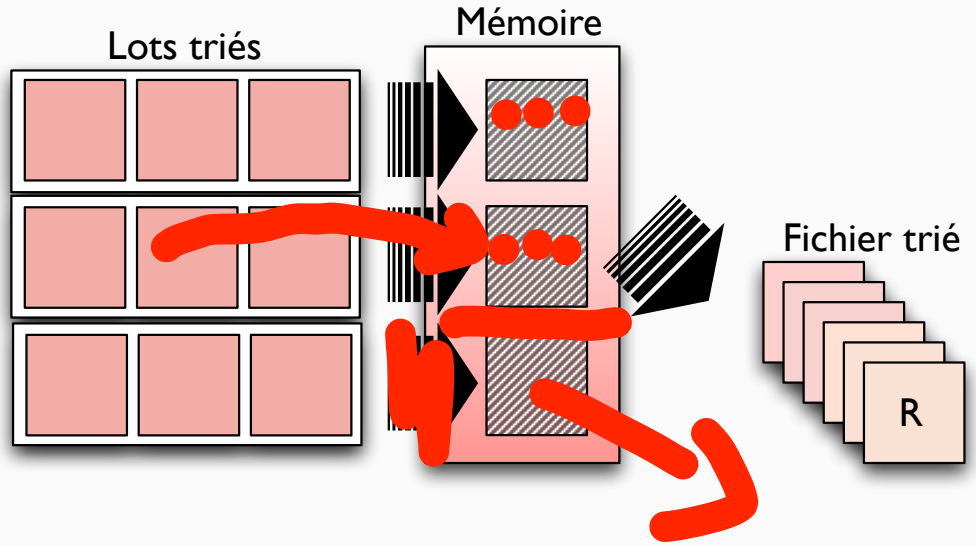
Une façon de trier : le tri-fusion (externe)

- Étape n°1 – Pour chaque lot de 100 pages de R :
  1. Lire le lot
  2. Trier les pages en mémoire
  3. Écrire les pages sur le disque



Épisode suivant

- Étape n°2 –
  1. Lire la première page de chaque lot trié
  2. Fusionner en fonction de l'ordre des valeurs
  3. Écrire le résultat sur disque
  4. Lire les pages suivantes et répéter l'opération jusqu'à épuisement de tous les lots



À quel prix ?

Calcul de coût

Chaque n-uplet est lu puis écrit (phase 1), lu puis écrit (phase 2)

- Coût du tri de R_1 : $4 \cdot 1\,000 = 4\,000$ I/Os
- Coût du tri de R_2 : $4 \cdot 500 = 2\,000$ I/Os

Calcul valable sous quelle hypothèse ?

Retour sur la jointure par fusion

Exemple 1(d) – suite

- Rappel des hypothèses : R_1 et R_2 contiguës mais non triées
- Coût total = coût de tri + coût de jointure (fusion)

$$\text{Coût total} = (4\,000 + 2\,000) + 1\,500 = 7\,500 \text{ I/Os}$$

5 500 (Itération)

Une question d'échelle

Sur de plus grandes relations

Soient R_1 pesant 10 000 pages ; R_2 à 5 000 pages ;
R_1 et R_2 contiguës et non triées

- Itération

$$\frac{5\,000}{100} \cdot (100 + 10\,000) = 505\,000 \text{ I/Os}$$

- Fusion

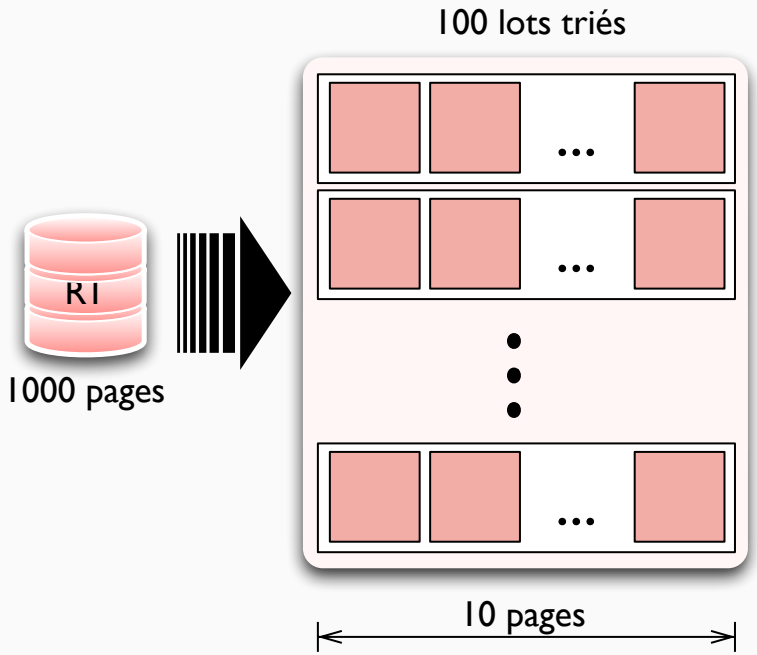
$$5 \cdot (10\,000 + 5\,000) = 75\,000 \text{ I/Os}$$

$\rightarrow$ La fusion (avec le tri) devient meilleure !

L'empreinte du tri

Quel est le volume de mémoire nécessaire au tri-fusion ?

Supposons que nous disposions de 10 pages de mémoire ;



→ la fusion nécessite 100 pages !

Étude du cas général

Soient k = # pages de mémoire ; x = # pages de la relation à trier ;

On dérive alors :

- # lots = x/k
- taille d'un lot = k

Contrainte à satisfaire : # lots $\leq$ # pages disponibles pour la fusion

$$x/k \leq k$$

$$k^2 \geq x \qquad k \geq \sqrt{x}$$

Détour par l'exemple

Ce qui donne sur R_1 et R_2 :

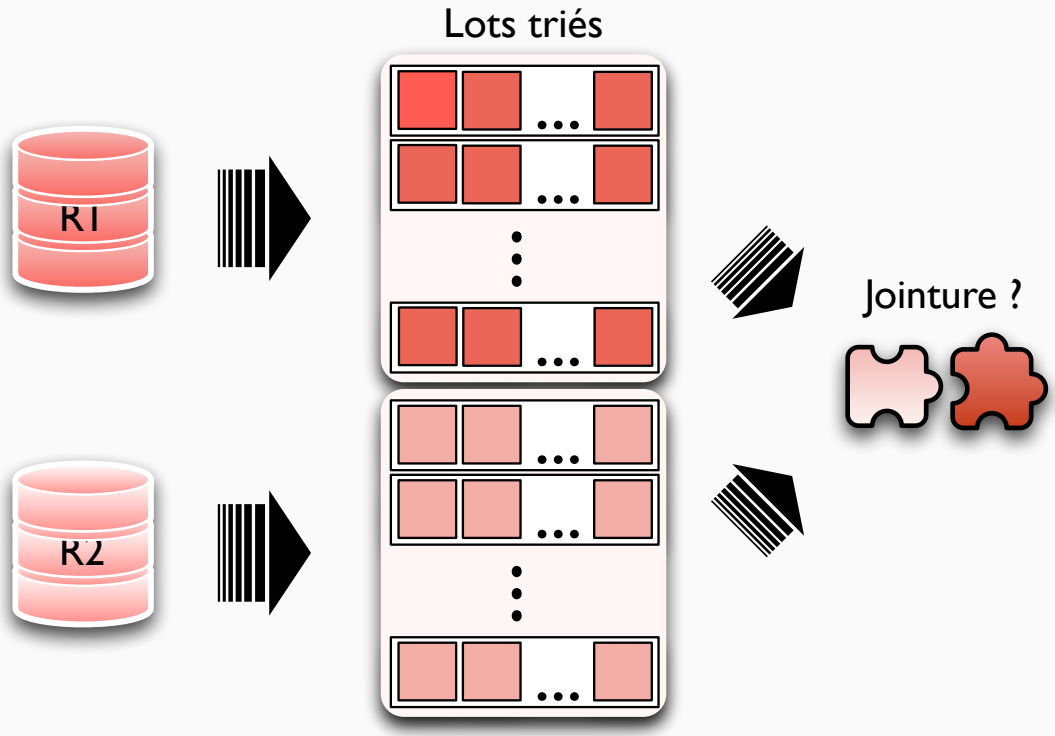
- R_1 fait 1.000 pages, donc $k \geq 31,62$
- R_2 fait 500 pages, donc $k \geq 22,36$

Il faut au moins 32 pages de *buffer*

Note : et si l'on n'a pas 32 pages ?

Amélioration de la jointure par fusion

Astuce A-t'on vraiment besoin d'un fichier complètement trié ?



Pour un prix de…

Coût de la jointure par fusion améliorée

Analyse :

- Lecture de R_1 + écriture de R_1 par lots
- Lecture de R_2 + écriture de R_2 par lots
- Jointure par fusion

$$\text{Coût total} = 2\,000 + 1\,000 + 1\,500 = 4\,500 \text{ l/Os}$$

Quelle est la quantité de mémoire nécessaire ?

Exemple 1(e) – Jointure avec index

Hypothèses

- Il existe un index sur $R_1.C$ comprenant 2 niveaux
- R_2 est contiguë et non triée
- L'index sur $R_1.C$ tient intégralement en mémoire

Petits calculs entre amis

Coût

- Lecture de R_2 : 500 I/Os
- Pour chaque n-uplet t_2 de R_2 :
  1. Parcourir l'index avec la clé $t_2.C$ // gratuit
  2. Si la valeur de $t_2.C$ existe, lire le n-uplet de R_1 // 1 I/O

Estimation

Combien y-a-t-il de bons n-uplets de R_1 ?

(a) $R_1.C$ est clé primaire et $R_2.C$ clé étrangère ;
n-uplets = 1

(b) $\mathsf{V}(R_1, C) = 5\,000$ et $\mathsf{T}(R_1) = 10\,000$, avec l'hypothèse de *répartition uniforme* ;
n-uplets = $\frac{10\,000}{5\,000} = 2$

(c) $\mathrm{dom}(R_1, C) = 1\,000\,000$ et $\mathsf{T}(R_1) = 10\,000$, avec l'hypothèse alternative ;
n-uplets = $\frac{10\,000}{1\,000\,000} = \frac{1}{100}$

Au final

Coût total de la jointure avec index

(a) Coût total $= 500 + 5\,000 \cdot (1) \cdot 1 = 5\,500$

(b) Coût total $= 500 + 5\,000 \cdot (2) \cdot 1 = 10\,500$

(c) Coût total $= 500 + 5\,000 \cdot (\frac{1}{100}) \cdot 1 = 550$

Plus subtil

Que se passe-t-il si l'index ne tient pas en mémoire ?

Exemple

L'index sur $R_1.C$ pèse 201 pages

- La racine + 99 feuilles chargés en mémoire
- Coût estimé de chaque parcours d'index :

$$E = (0) \cdot \frac{99}{200} + (1) \cdot \frac{101}{200} \approx 0,5$$

Calcul de coût avec parcours d'index

$$\begin{aligned}\text{Coût total} &= 500 + 5\,000 \cdot [\text{parcours} + \text{lecture n-uplet}] \\ &= 500 + 5\,000 \cdot [0,5 + 2] \qquad \text{// répartition uniforme} \\ &= 500 + 12\,500 = 13\,000 \text{ I/Os} \qquad \text{// cas (b)}\end{aligned}$$

Et dans le cas (c) :

$$\begin{aligned}\text{Coût total} &= 500 + 5\,000 \cdot [0,5 + \frac{1}{100} \cdot 1] \\ &= 500 + 5\,000 + 50 = 3\,050 \text{ I/Os}\end{aligned}$$

Bilan provisoire

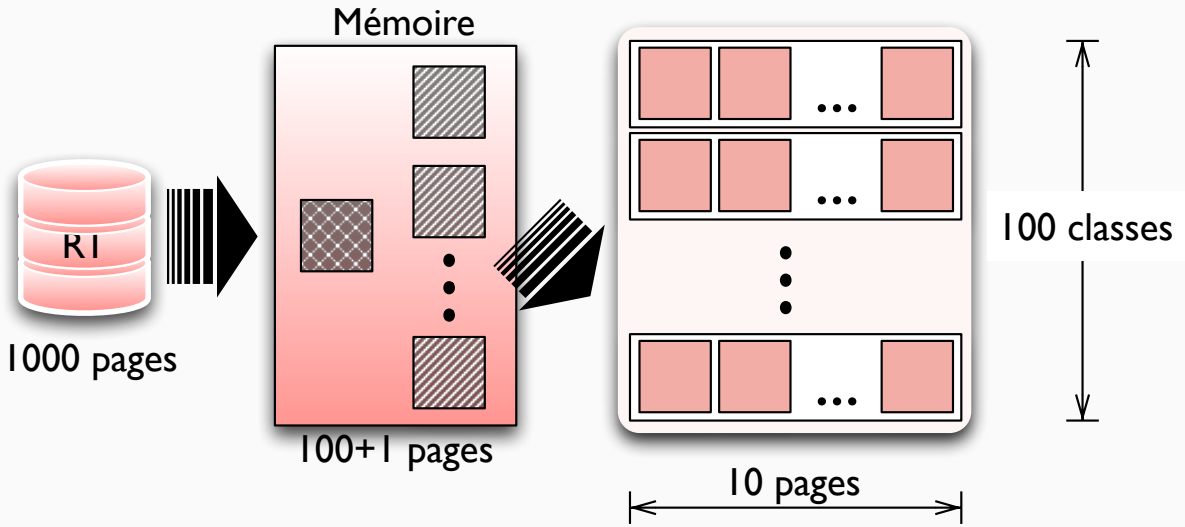
Coût des algorithmes de jointure

- Relations non contiguës
  - Itération $R_2 \bowtie R_1$ ..... 505 000 (meilleur)
  - Fusion ..... —
  - Tri+fusion ..... —
  - Index sur $R_1.C$ ..... —
  - Index sur $R_2.C$ ..... —
- Relations contiguës
  - Itération $R_2 \bowtie R_1$ 5 500
  - Fusion 1 500
  - Tri+fusion ..... 7 500→4 500
  - Index sur $R_1.C$ ..... 5 500→3 050→550
  - Index sur $R_2.C$ ..... —

Exemple 1(f) – Jointure par hachage

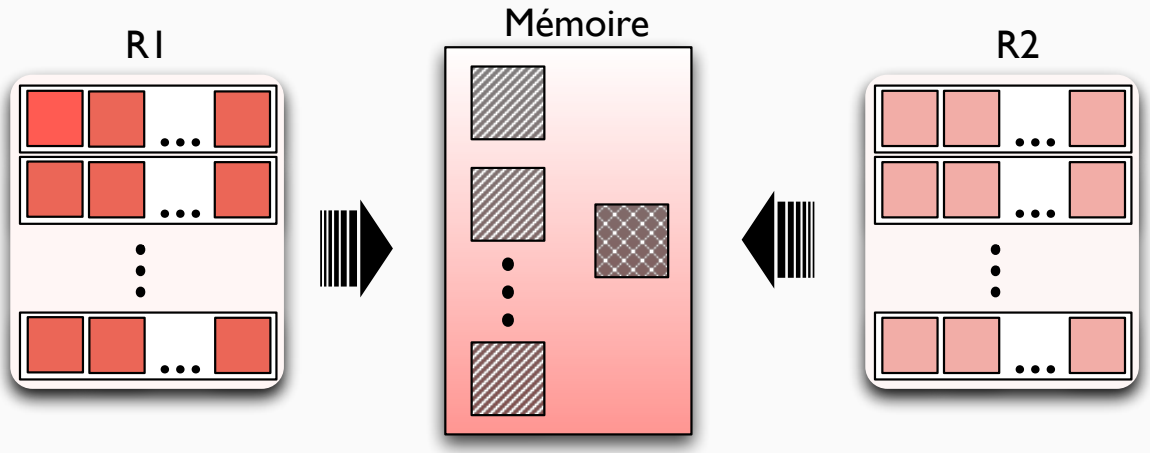
Hypothèses R_1 et R_2 contiguës (et non triées)

1. Prévoir 100 classes (*buckets*)
2. Lire R_1, hacher, puis écrire chaque classe



Suite de la jointure par hachage

3. Id. pour R_2
4. Lire une classe de R_1 ; Construire H-table en mémoire
5. Lire la classe équiv. de R_2 + Recherche par H-code



Encore des entrées/sorties

Calcul de coût pour la jointure par hachage

- Construction des classes :
  - Lecture de R_1 + Écriture
  - Lecture de R_2 + Écriture
- Jointure :
  - Lecture de R_1 et R_2

$$\text{Coût total} = 3 \cdot (1\,000 + 500) = 4\,500 \text{ I/Os}$$

Remarque : ce n'est qu'une approximation car la taille des classes varie et on arrondit systématiquement à "la page" supérieure

Paramétrage

Quelle quantité de mémoire pour la jointure par hachage ?

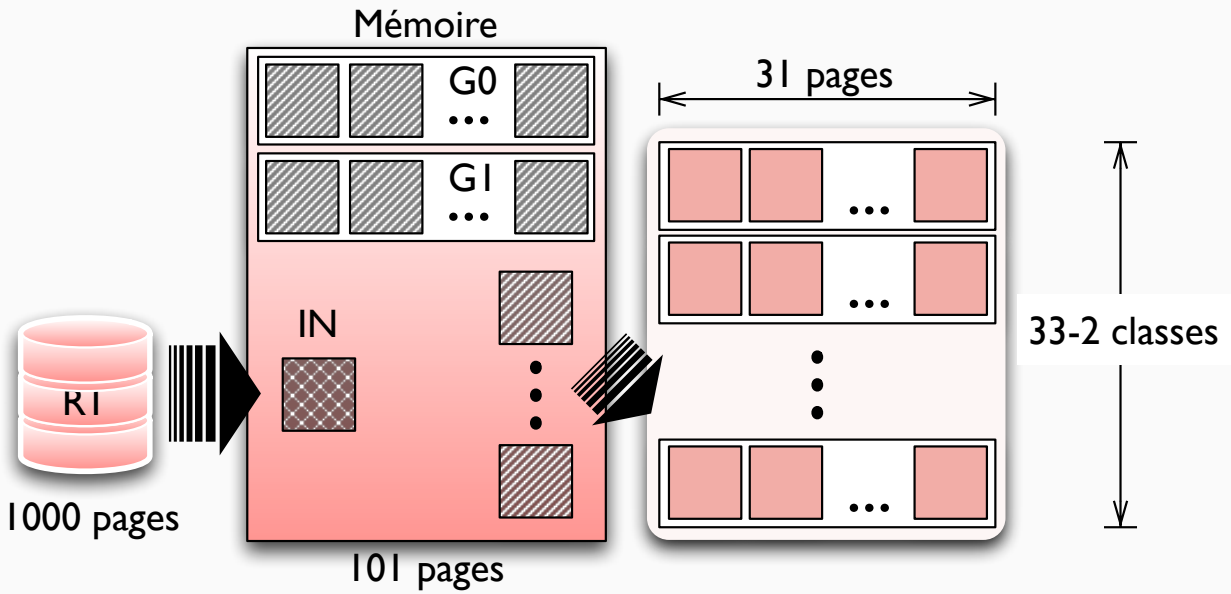
- Taille d'une classe : (x/k)
  - k : # pages mémoire
  - x : # pages de R_1
- Donc $(x/k) < k$, ou encore $k > \sqrt{x}$

En fait, il faut $k+1$ pages mémoire de *buffer* pour l'opération de jointure avec R_2

Amélioration de la jointure par hachage

Astuce Conserver des classes en mémoire

Par exemple, $k = 33$; chaque classe de R_1 fait 31 pages ; on conserve 2 classes en mémoire



Détails

Usage de la mémoire

- Classe G_0 . . . . . 31 pages
- Classe G_1 . . . . . 31 pages
- Sortie (autres classes) . . . . . 33-2 pages
- Entrée (lecture de R_1) . . . . . 1 page

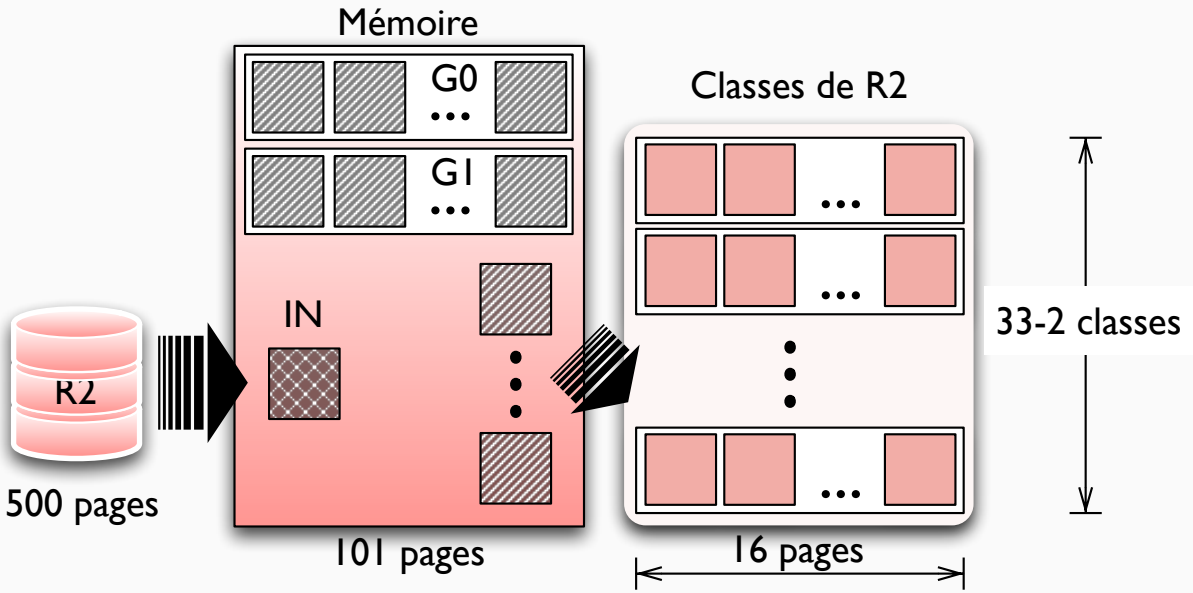
$$\text{Total} = 94 \text{ pages}$$

Il reste 7 pages non consommées !

Suite du hachage hybride

Ensuite : Hacher R_2

- Classes de R_2 = 500/33 = 16 pages
- 2 des classes sont jointes instantanément avec G_0 et G_1

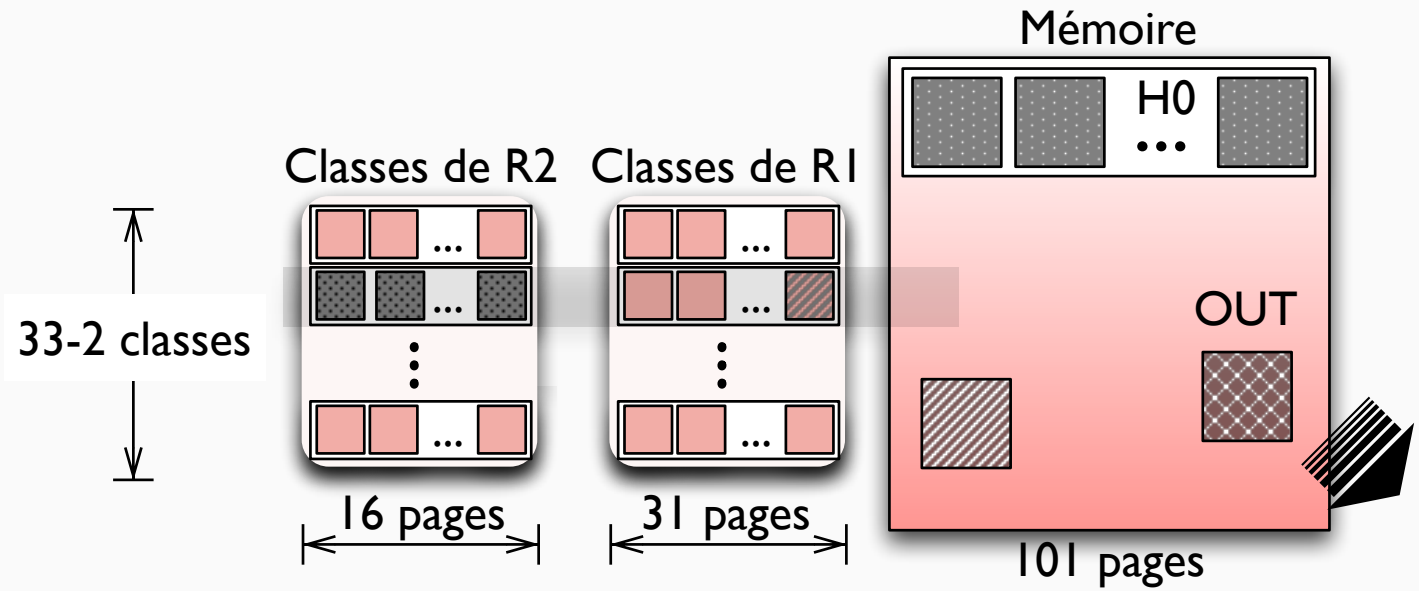


Fin du hachage hybride

Joindre les classes restantes

Pour chaque couple de classes :

- Lire une des classes en mémoire
- joindre en parcourant la seconde



À quel prix ?

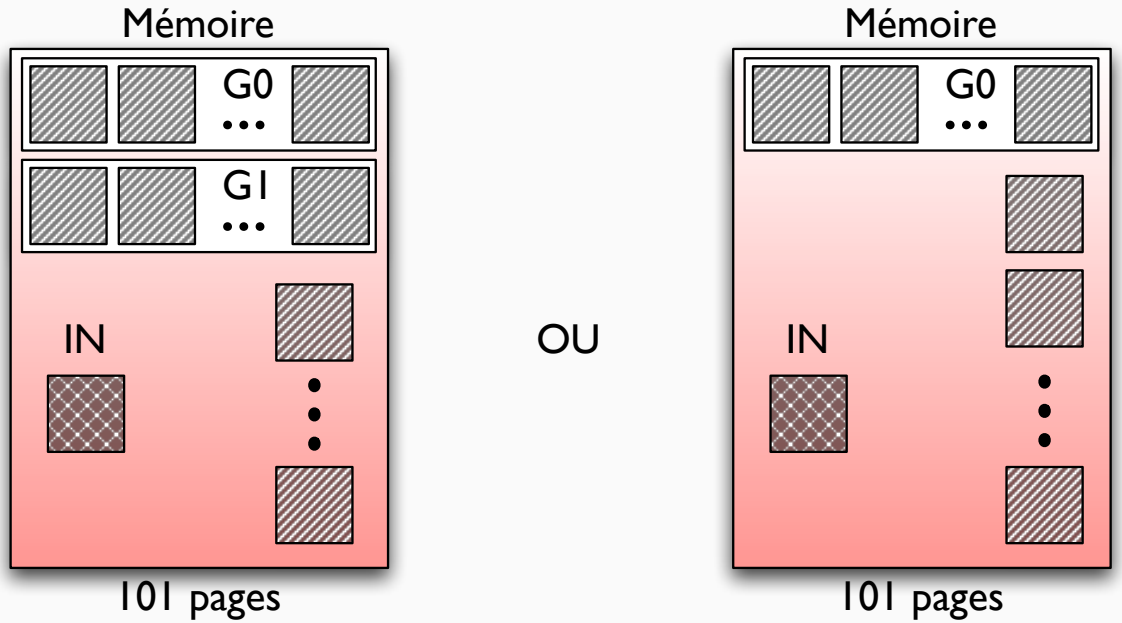
Calcul de coût pour la jointure par hachage hybride

- Hachage de $R_1 = 1\,000 + 31 \cdot 31 = 1\,961$ I/Os
- Pour hacher R_2, il suffit d'écrire 31 classes
  Coût $= 500 + 31 \cdot 16 = 996$ I/Os
- Pour réaliser les jointures manquantes :
  Coût $= 31 \cdot 31 + 31 \cdot 16 = 1\,457$ lectures

$$\text{Coût total} = 1\,961 + 996 + 1\,457 = 4\,414 \text{ I/Os}$$

Question intéressante

Combien faut-il conserver de classes en mémoire ?



Pour aller *encore* plus loin

Une autre astuce pour la jointure par hachage

- N'écrire que les couples $\langle v, @ \rangle$ dans les classes
- Au moment de la jointure, si les valeurs correspondent, lire les deux n-uplets

Par les couples valeur/adresse

Pour illustrer le calcul de coût

Hypothèses

- 100 couples $\langle v, @ \rangle$ par page
- 100 n-uplets attendus dans le résultat

1. Construire une H-table pour R_2 en mémoire
   5.000 n-uplets $\rightarrow 5\,000/100 = 50$ pages
2. Lire R_1 et comparer les valeurs
3. Lire environ 100 n-uplets de R_2

Coût total $=$ Lecture de $R_2 +$ Lecture de $R_1 +$ Accès n-uplets de R_2

$= 500 + 1\,000 + 100 = 1\,600$ I/Os

Bilan définitif

Pour des relations contiguës

- Itération 5 500
- Fusion 1 500
- Tri+fusion 7 500
- Index sur $R_1.C$ ..... 5 500→550
- Index sur $R_2.C$ ..... —
- Construction de l'index sur $R_1.C$ ..... —
- Construction de l'index sur $R_2.C$ ..... —
- Hachage ..... 4 500+
- Hachage avec astuce, d'abord R_1 ..... 4 414
- Hachage avec astuce, d'abord R_2 ..... —
- Hachage avec pointeurs 1 600

C’est la fin

En résumé

- Itération ok pour petites relations (du point de vue de la capacité mémoire)
- Pour les équi-jointures, lorsque les relations ne sont pas triées et qu’il n’existe aucun index, la jointure par hachage est généralement la meilleure
- Tri + jointure par fusion est recommandé pour les autres prédicats (par ex. $R_1.C > R_2.C$)
- Si les relations sont triées, utiliser la fusion
- S’il existe un index, il peut être intéressant de l’utiliser (cela dépend de la taille estimée du résultat)

Reste à faire

Ce qui n'a pas formellement été dit (entre autre)

- Gestionnaire de mémoire
- Algorithmes et coût pour les autres opérateurs algébriques
- Algorithmes à itérations multiples (> 2)
- Algorithmes parallèles
- Optimisation de requêtes réparties
- ...

Tuning

Le réglage (tuning) des bases de données

Une promenade au niveau de l'organisation « physique » de la base de données

Sujets à traiter

- Une définition du problème
- Le choix des indexes et des vues matérialisées
- L'écriture des requêtes
- La fragmentation verticale et horizontale
- La « dé-normalisation »

L’énoncé du problème

Étant donné le profil de *charge de travail* :

- La liste des requêtes avec leurs fréquences
- La liste des écritures avec leurs fréquences
- Des critères de performance pour chaque type de transaction

Le tuning de base de données

C’est le réglage des détails d’implémentation dans le but d’atteindre la performance souhaitée pour le traitement des données, relativement au profil de charge

F.A.Q. à propos du profil de charge

- Quelles tables sont concernées ?
- Quels attributs au sein de ces tables sont-ils examinés ?
- Quels sont ceux qui sont impliqués dans des critères de sélection ou de jointure ?
- Quelle est la sélectivité de ces critères ?

Réponse à formuler pour chaque requête de la charge de travail

Du choix des indexes

Étant donné un schéma de base de données ;
étant donné le profil de charge de travail ;

- un ensemble de paires (transaction, fréquence), avec
- la transaction en lecture (`SELECT`) ou en écriture (`INSERT`/`UPDATE`/`DELETE`)
- la fréquence en valeur brute ou en proportion

Objectif

Déterminer l'ensemble minimal d'indexes qui maximise la performance de la charge de travail

C'est *a priori* un problème difficile

La clé d'index

Un index est construit sur un attribut K dès lors que la clause `WHERE` contient :

- une condition d'égalité sur K, ou
- un test d'intervalle sur K, ou
- une jointure sur K, également formulée avec la clause `JOIN`

Les indexes : cas numéro 1

Soit le schéma $R(A, B, C)$;
et le profil de charge suivant :

100 000 requêtes :

```
SELECT *
FROM R
WHERE A=?
```

100 requêtes :

```
SELECT *
FROM R
WHERE C=?
```

Quels indexes ?

Les indexes : cas numéro 1

Soit le schéma $R(A, B, C)$;
et le profil de charge suivant :

100 000 requêtes :

```
SELECT *
FROM R
WHERE A=?
```

100 requêtes :

```
SELECT *
FROM R
WHERE C=?
```

Quels indexes ?

$R(A)$ et $R(C)$, par hachage ou arbre B

Les indexes : cas numéro 2

Soit le schéma $R(A, B, C)$;
et le profil de charge suivant :

100 000 requêtes :

```
SELECT *
FROM R
WHERE A>? AND A<?
```

100 requêtes :

```
SELECT *
FROM R
WHERE C=?
```

100 000 requêtes :

```
INSERT INTO R
VALUES (?,?,?)
```

Quels indexes ?

Les indexes : cas numéro 2

Soit le schéma $R(A, B, C)$;
et le profil de charge suivant :

100 000 requêtes :

```
SELECT *
FROM R
WHERE A>? AND A<?
```

100 requêtes :

```
SELECT *
FROM R
WHERE C=?
```

100 000 requêtes :

```
INSERT INTO R
VALUES (?,?,?)
```

Quels indexes ?

impérativement un arbre B sur $R(A)$, et probablement rien sur $R(C)$

Les indexes : cas numéro 3

Soit le schéma $R(A, B, C)$;
et le profil de charge suivant :

100 000 requêtes :

```
SELECT *
FROM R
WHERE A=?
```

1 000 000 requêtes :

```
SELECT *
FROM R
WHERE A=? AND C>?
```

100 000 requêtes :

```
INSERT INTO R
VALUES (?,?,?)
```

Quels indexes ?

Les indexes : cas numéro 3

Soit le schéma $R(A, B, C)$;
et le profil de charge suivant :

100 000 requêtes :

```
SELECT *
FROM R
WHERE A=?
```

1 000 000 requêtes :

```
SELECT *
FROM R
WHERE A=? AND C>?
```

100 000 requêtes :

```
INSERT INTO R
VALUES (?,?,?)
```

Quels indexes ?

$R(A, C)$—et non $R(C, A)$

Les indexes : cas numéro 4

Soit le schéma $R(A, B, C)$;
et le profil de charge suivant :

1 000 requêtes :

```
SELECT *
FROM R
WHERE A>? AND A<?
```

100 000 requêtes :

```
SELECT *
FROM R
WHERE C>? AND C<?
```

Quels indexes ?

Les indexes : cas numéro 4

Soit le schéma $R(A, B, C)$;
et le profil de charge suivant :

1 000 requêtes :

```
SELECT *
FROM R
WHERE A>? AND A<?
```

100 000 requêtes :

```
SELECT *
FROM R
WHERE C>? AND C<?
```

Quels indexes ?

$R(C)$ plaçant—primaire—et $R(A)$ non plaçant—secondaire

Principes directeurs

Pour choisir les indexes

- traiter les transactions par ordre d'importance décroissante
- Se concentrer uniquement sur les tables sollicitées par les transactions
- Examiner les clauses `WHERE` pour trouver d'éventuelles clés d'index
- Privilégier les indexes à même d'accélérer plusieurs requêtes
- Ou encore, tenir compte de ce qui est dit dans la suite…

Les clés composées

Conditions pour un index composite sur $(K_1, \ldots, K_n)$

- Une clause `WHERE` porte sur la combinaison K_1, ..., K_n
- Une clause `SELECT` contient uniquement K_1, ..., K_n

Index couvrant

Il permet de donner la réponse à une requête, sans accès aux données elles-mêmes

Example (Index couvrant sur (K_1, K_2))

```
SELECT K2 FROM R WHERE K1=55
```

Index plaçant ou non ?

- Les requêtes d'intervalle tirent profit du groupement des données selon la clé de recherche
- Les indexes couvrants sont insensibles au groupement : ils fonctionnement parfaitement avec ou sans
- les prédicats d'égalité ($X = v$) sont aussi de bons candidats pour envisager un index plaçant, voire un groupement par hachage

Pour fixer les idées

Example (Requête d'intervalle)

```
SELECT * FROM R WHERE K>? AND K<?
```

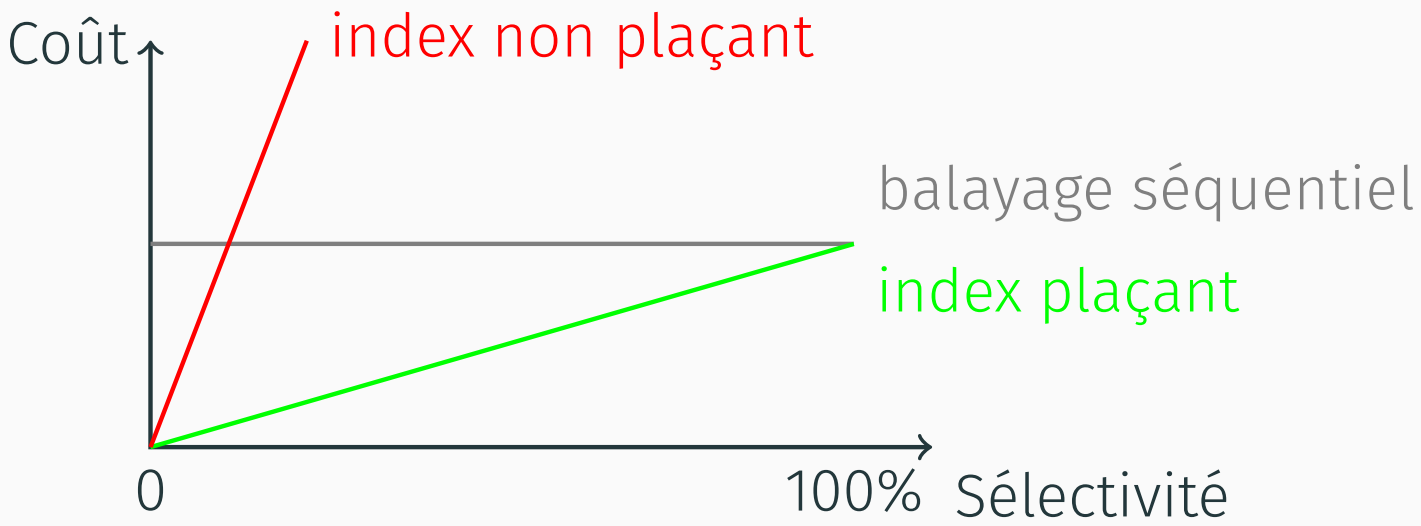


Table de hachage vs. arbre B+

Règle n°1

Toujours construire un arbre B+ ☺

Règle n°2

Envisager une table de hachage sur K si :

- il existe une requête importante avec un prédicat d'égalité (`WHERE K=?`) et aucune requête d'intervalle sur K, ou
- vous savez que l'optimiseur réalise une jointure par itérations pour laquelle K est l'attribut de jointure de la relation interne (*rhs* ou « à droite »)

Équilibrer requêtes et écritures

- Les indexes accélèrent les requêtes
  - `SELECT FROM WHERE`
- Mais la plupart du temps, ils pénalisent les écritures
  - `INSERT`, `DELETE`, `UPDATE`

Néanmoins, certaines écritures tirent profit des indexes !

Example

```
UPDATE R SET A=7 WHERE K=55
```

Les vues matérialisées

Le concept

Pré-calcul et stockage d'une requête sous la forme d'une table

Example

```
CREATE MATERIALIZED VIEW mv_agr AS
    ( SELECT P.nom AS nom, AVG(P.prix) AS prix
    FROM Produit P NATURAL JOIN Commande C
    GROUP BY P.nom HAVING COUNT(*)>10 )
```

Usage

```
SELECT DISTINCT M.nom FROM mv_agr M, Produit P
WHERE M.nom=P.nom AND M.prix<P.prix
```

Scénarios possibles

Il existe une vue matérialisée $V = R \bowtie S$

Requête $Q = R \bowtie S \bowtie T$

- Peut être remplacée par $Q = V \bowtie T$
- En fonction de l'estimation du coût

Requête $\sigma_{A=a}(V)$

- Il existe un index sur $R.A$, et
- un autre sur l'attribut de jointure $S.B$
- Le plan avantageux : $Q = \sigma_{A=a}(R) \bowtie S$

Les vues matérialisées : Pros & Cons

Avantages

- Définition aisée
- Accélération des requêtes coûteuses
  - Production instantanée du résultat
- Réglages au même titre qu'une table

Inconvénients

- Mise à jour !
  - réévaluation périodique intégrale, ou
  - maintenance incrémentale à définir à la main

Performant pour des requêtes coûteuses sur données stables

La réécriture de requête SQL

Tâche complexe

Interaction de :

- valeurs `NULL`
- doublons
- agrégation
- requêtes imbriquées

Bonne nouvelle

L'optimiseur fait une bonne part du boulot !

Huggy les bons tuyaux 1/3

Idée n°1

1 seul bloc d'optimisation partout où c'est a priori possible

Example

requête corrélée

```
SELECT E.nom FROM Enseignant E
WHERE EXISTS ( SELECT * FROM Enseigne N
                 WHERE E.id=N.id AND N.annee=2016 )
```

requête aplatie

```
SELECT E.nom FROM Enseignant E, Enseigne N
WHERE E.id=N.id AND N.annee=2016
```

Et avec un `NOT EXISTS` ?

Huggy les bons tuyaux 2/3

Idée n°2

Limiter l'usage du `DISTINCT`, notamment si le résultat contient une clé

Idée n°3

Limiter l'usage de la construction `HAVING` avec le `GROUP BY`

Huggy les bons tuyaux 3/3

Idée n°4

Proscrire la création de tables temporaires

Idée n°5

Écrire des blocs d'union plutôt que des conditions `OR` dans la clause `WHERE`

...

Révision du schéma

Examen des formes normales alternatives du schéma

Rappel

Un schéma admet souvent plusieurs FNBC/3FN

Bricolages supplémentaires

- Fragmentation horizontale ou verticale
- Dé-normalisation : dégrader la forme normale de la BdD
- Réplication d'attribut

Fragmentation verticale

Curriculum=

NSS	Nom	Adresse	Cv	Photo
123123	Alice	Nantes	Clob1...	Blob1...
234234	Bob	Paris	Clob2...	Blob2...
345345	Carole	Lyon	Clob3...	Blob3...
456456	David	Nantes	Clob4...	Blob4...

▼

T_1

NSS	Nom	Adresse
123123	Alice	Nantes
234234	Bob	Paris
...	...	...

T_2

NSS	Cv
123123	Clob1...
234234	Clob2...

T_3

NSS	Photo
123123	Blob1...
234234	Blob2...

Sur-décomposition au-delà de la FNBC/3FN !

Que devient la table originale ?

Besoin de préserver la table Cv pour certaines applications/requêtes ?

```
CREATE VIEW Cv AS
    SELECT T1.NSS, T1.Nom, T1.Adresse,
           T2.Cv,
           T3.Photo
    FROM T1
         NATURAL JOIN T2
         NATURAL JOIN T3
```

Du bon usage de la fragmentation verticale

Example

```
SELECT Adresse
FROM Cv
WHERE Nom='Alice'
```

- Quelles sont les tables parmi T_1, T_2 et T_3 qui sont concernées par la requête ?
- Quand doit-on utiliser la fragmentation verticale ?

Les motivations 1/2

1. La performance des requêtes

Lorsque peu de colonnes sont affectées :

- Accès disque uniquement pour ces colonnes
- économie I/O substantielle pour des tables « larges »
- Utile pour les entrepôts multidimensionnels

Inconvénients

- Surcoût de stockage pour les répliques de la clé
- Jointures coûteuses pour reconstruire les n-uplets

Les motivations 2/2

2. Les « gros » attributs rarement sollicités
   - texte long, document
   - image, son
   - *etc.*

3. Les bases de données réparties
   - Infos personnelles sur un site,
   - activité et profil sur un autre

4. L'intégration de données
   - T_1 provient d'une source
   - T_2 d'une autre

Fragmentation horizontale

Client=

NSS	Nom	Ville	Dpt
123123	Alice	Nantes	44
234234	Bob	Paris	75
345345	Carole	Lyon	69
456456	David	Rezé	44
567567	Eva	Paris	75
678678	Franck	Nantes	44

ClientParis=

NSS	Nom	Ville	Dpt
234234	Bob	Paris	75
567567	Eva	Paris	75

ClientLyon=

NSS	Nom	Ville	Dpt
345345	Carole	Lyon	69

Client44=

NSS	Nom	Ville	Dpt
123123	Alice	Nantes	44
456456	David	Rezé	44
678678	Franck	Nantes	44

Recomposition

```
CREATE VIEW Client AS
    ( ClientParis
        UNION ALL
      ClientLyon
        UNION ALL
      Client44 )
```

Et usage

Example

```
SELECT Nom
FROM Clients
WHERE Ville='Paris'
```

Quelles sont les tables qui sont sollicitées ?!

Un peu mieux

```
CREATE VIEW Client AS
    ( SELECT * FROM ClientsParis WHERE Ville='Paris' )
        UNION ALL
    ( SELECT * FROM CLientsLyon WHERE Ville='Lyon' )
        UNION ALL
    ( SELECT * FROM Clients44 WHERE Dpt='44' OR
                                    Ville='Nantes' OR
                                    Ville='Rezé' )
```

- Besoin de « marqueur » de fragmentation
- Techniques alternatives en fonction du SGBD

Retour à l'usage

```
SELECT Nom
FROM Client
WHERE Ville='Paris'
```

▼

```
SELECT Nom
FROM ClientParis
```

Motivations pour la fragmentation horizontale

1. Performance
    - Particulièrement pour les entrepôts
        - 1 fragment par mois
        - données historisées et données actives
        - *etc.*
2. BdD réparties et parallèles
3. Intégration de données

Révision du schéma

- Fragmentation
- **Dé-normalisation**

Les variantes de la dé-normalisation 1/2

1. Concrétiser un attribut calculé
   - `Disque(titre, dateSortie, label, durée)`
   - somme des durées de chaque piste
2. Ré-introduire la transitivité
   - `Employé(nom, fonction, batId, dptId)`
   - dépendance fonctionnelle `batId` $\rightarrow$ `dptId`
3. Promouvoir un attribut
   - `Employé(nom, fId, batId)` et `Fonction(fId, fNom)`
   - `fonction` comporte peu de modalités de grande taille

Les variantes de la dé-normalisation 2/2

4. Fusionner des tables
   - `Employé(nom, fonction, batId, bNom, bAdresse)`
   - pré-calcul de la jointure $\texttt{Employé} \bowtie \texttt{Bâtiment}$

5. Dupliquer un attribut
   - `Employé(nom, fonction, batId, bNom)`
   - forme faible de la fusion

Étude de cas

Example (Le schéma)

```
Produit(pId, pNom, prix, fId)
Fournisseur(fId, fNom, ville)
```

Une requête très fréquente

```
SELECT P.pId, P.pNom
FROM Produit P
     NATURAL JOIN Fournisseur F
WHERE P.prix<? AND F.ville=?
```

- Quelles optimisations ?

Révision du schéma

Rappel

```
Produit(pId, pNom, prix, fId)
Fournisseur(fId, fNom, ville)
```

Dé-normalisation

```
ProduitFournisseur(pId, pNom, prix, fNom,
ville)
```

Retour à la requête d'exemple

```
SELECT P.pId, P.pNom
FROM Produit P
     NATURAL JOIN Fournisseur F
WHERE P.prix<? AND F.ville=?
```

▼

```
SELECT pId, pNom
FROM ProduitFournisseur
WHERE prix<? AND ville=?
```

Les problèmes de la dé-normalisation

- entorse à la FNBC/3FN
  - fId $\rightarrow$ fNom, ville
- redondance
  - sur-poids
  - mise à jour de toutes les répliques d'une valeur
- anomalies
  - mise en œuvre de contrôles manuels (*trigger*, appli.)
- requêtes ciblées moins efficaces
  - la table dé-normalisée est plus coûteuse à balayer

Dé-normaliser ou pas ?

Redondance volontaire justifiée sous conditions

1. la performance est un objectif et la redondance y contribue
2. les DF responsables de la dé-normalisation sont documentées
3. les mécanismes de contrôle sont en place

Aperçu de la complexité du réglage

Pour accélérer une jointure fréquente :

- création d'indexes
- définition d'une vue matérialisée
- dé-normalisation
- groupement des tables !

Réglage d'une requête complexe

Il arrive qu'une seule requête concentre les efforts de tuning

Pour régler une requête

1. Examiner le plan d'exécution de l'optimiseur
   - `EXPLAIN PLAN`
   - relever les indexes, les algorithmes de jointure, les méthodes d'accès, *etc.*
2. Proposer de nouveaux indexes et vues matérialisées
3. Réviser le schéma de la base de données
4. Reformuler la requête

Effet secondaire Attention à ne pas pénaliser les autres transactions !