INSIA
Bases de données
Optimisation – 2 : arbres algébriques et calcul de coût

Bertrand LIAUDET

SOMMAIRE

SOMMAIRE 1

INTRODUCTION 3

Bibliographie 3

Généralités sur l'optimisation des requêtes 3

Objectif de l'optimisation 3

Typologie des requêtes d'un point de vue général d'optimisation 3

L'optimiseur et optimisation 4

3 approches de l'optimisation 5

Optimisation physique et notion de METABASE 6

Plan d'exécution 6

Bilan de l'optimisation : le plan d'exécution choisi et calcul de coût 6

Optimisation et méthodes (algorithmes) d'accès aux données 7

La question de la performance : l'évolution des SGBD en terme de rapidité 7

OPTIMISATION SYNTAXIQUE 8

1. L'arbre algébrique 8

Exemple traité 8

Rappel de vocabulaire 9

Représentation textuelle des opérateurs de l'algèbre relationnelle 10

Représentation graphique des opérateurs de l'algèbre relationnelle dans les arbres algébriques 10

Arbre algébrique 11

Notion d'arbres algébriques équivalent 12

2. Théorie sur les règles de transformation des arbres algébriques 14

Règles de base sur les produits cartésiens et les jointures 14

Règles de base sur les restrictions 14

Règles de base sur les projections 15

3. Algorithme d'optimisation syntaxique et arbres algébriques optimisés 17

Algorithme d'optimisation syntaxique 17
Arbres algébriques optimisés 17

CALCUL DE COUT ET OPTIMISATION PHYSIQUE 20

1. Première approche du calcul du coût : approche SQL 20
Quantité de données 20
Principes du calcul 21
Exemple 21

2. Optimisation phyisique et calcul de coût indexé 23
Principe du passage du SQL au langage impératif 23
La jointure indexée 23
Distinction entre jointure indexée unique (EQ_REF) et jointure indexé multiple (REF) 24
Exemple 24
Pourquoi choisir une solution plutôt qu'une autre ? 26
Calcul de coût et arbre algébrique indexé 26
Les restrictions dans les arbres algébriques indexés 27

EXERCICES 29

1. Les employés 29
2. La bibliothèque 30
3. Les projets 31
Schéma de la BD projets simplifié 31
4. La maison de disques 32
Schéma de la BD Maison de disques 32
5. L'école 33
Schéma de la BD Ecole 33

Première édition : février 2008
Deuxième édition : septembre 2009
Troisième édition : avril 2010

INTRODUCTION

Bibliographie

Gardarin. Bases de données. Eyrolles 2005. §10 – pp. 301 à 350.

Darmaillac, Rigaux. Maîtriser MySQL 5. O'Reilly 2005. §11 – pp. 283 à 318.

Dubois, Hinz, Pedersen. MySQL-Guide officiel. Campus press 2004. §13 – pp. 433 à 475.

MySQL 5-Guide de l'administrateur. Campus press 2005. §6 – pp. 447 à 510.

Harrison. MySQL Stored Procedure. O'Reilly 2006. Part IV – pp. 421 à 582.

Généralités sur l'optimisation des requêtes

Objectif de l'optimisation

L'objectif de l'optimisation est d'accélérer de vitesse de traitement des requêtes.

Typologie des requêtes d'un point de vue général d'optimisation

Informatique de gestion (au sens large) *vs.* informatique décisionnelle

Les requêtes de l'informatique de gestion (SI au sens large – informatique transactionnelle) concernent autant voir plus la création-modification-suppression (CMS : insert, update, delete) des données que leur interrogation. L'interrogation des données est souvent assez élémentaire.

Les requêtes de l'informatique décisionnelle (OLAP) concernent essentiellement l'interrogation des données. Ce sont des requêtes complexes, multi-tables, le plus souvent avec des statistiques et des agrégats, souvent écrites en PL-SQL.

Requêtes statiques (compilées) *vs.* requêtes *ad hoc* (interprétées, dynamiques)

Les requêtes statiques sont les requêtes programmées une fois pour toutes dans une application. Elles seront probablement utilisées plusieurs fois.

Les requêtes dynamiques sont les requêtes construites directement via une calculette SQL avec ou sans interface graphique : mysql sous MySQL, psql sous PostgreSQL, SQL-Plus sous Oracle, SQL Server Management Studio Express sous SQL-Server, etc. Elles sont probablement utilisées une seule fois.

Bilan

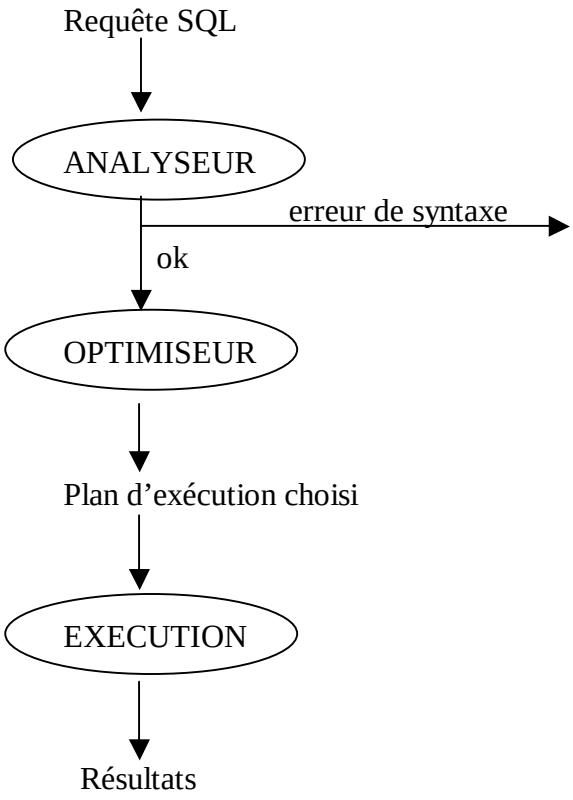
L'optimisation concerne **sutout les requêtes d'informatique transactionnelle** car l'usage intensif de l'informatique décisionnelle passe par des applications spécifiques qui complètent ou sont indépendante du SGBD.

L'optimisation concerne **surtout les requêtes statiques**. Le but est d'être le plus efficace possible, au moins pour les questions les plus fréquentes.

L'optimiseur et optimisation

L'optimiseur

L'optimiseur est un composant du SGBD. Son rôle est de transformer la requête SQL en opérations dites « de bas niveau » réalisant efficacement l'accès aux données.



En majuscule et dans un ovale : les traitements

En minuscules : les données

Principe général de l'optimisation des requêtes par l'optimiseur

L'optimisation consiste à passer d'une requête exprimée dans un **langage source** (le **SQL** dans le modèle relationnel) à une requête exprimée par un ensemble d'**opérations plus élémentaires** exprimées dans un **langage cible** (opérations dites « de bas niveau »).

Ces opérations élémentaires dépendent particulièrement :

- de la requête SQL initiale
- des contraintes d'intégrité
- des index
- des statistiques de la BD : taille des tables et connaissances sur les résultats des restrictions.

Les plans d'exécution

Le plan d'exécution d'une requête est une suite possible d'opérations de bas niveau qui permettent de réaliser cette requête.

Il peut exister plusieurs plans d'exécution pour une requête donnée.

C'est le résultat de l'optimisation.

Le plan d'exécution choisi et le calcul du coût

L'optimisation consiste à choisir le meilleur plan d'exécution (le plus perfomant). Le choix se fait par un calcul du coût (du temps de traitement).

Commande EXPLAIN

La commande EXPLAIN permet de consulter le plan d'exécution mis en œuvre par le SGBD pour une requête donné.

En général sa syntaxe se résume à : EXPLAIN requête ;

3 approches de l'optimisation

Il y a **3 approches** de l'optimisation des requêtes :

L'optimisation sémantique

- L'optimisation sémantique consiste essentiellement à rechercher une contradiction dans les restrictions qui conduirait à obtenir 0 tuple en réponse à la question ou à une partie de la question.

  La contradiction peut apparaître directement dans la question. C'est le cas si deux restrictions sont contradictoires : (degré > 13 et degré <10) ou encore (région = R1 et région = R2).

  La contradiction peut apparaître quand on croise une restriction avec une contrainte d'intégrité de la BD. Par exemple, une contrainte de valeur peut préciser : (degré <15) et une question demander (degré >15).

  Si une telle contradiction est découverte alors la table concernée ainsi que toutes celles qui lui sont jointes n'auront pas de tuples.
- L'optimisation sémantique peut aussi traiter certains points particuliers : par exemple remplacer un like par un « = » si possible (pas de caractères spéciaux dans la restriction).
- L'optimisation sémantique consiste donc à remplacer une formule par une autre à l'intérieur du select (une requête par la valeur null, un like par un « = », etc.) sans changer la structure de la requête.
- Cette optimisation est totalement mécanique.

L'optimisation syntaxique

- Elle prend uniquement en compte les **propriétés de l'algèbre relationnelle**. Autrement dit, l'ordre des opérations sera changé pour obtenir le meilleur résultat en terme de performance. Par exemple, une table sera utilisée avant une autre, une restriction sera faite avant une autre, une jointure sera faite avant une autre, etc.
- L'optimisation syntaxique consiste donc à réorganiser les étapes des différentes opérations relationnelles élémentaires d'une requête. Elle part d'une requête en SQL pour arriver à une autre requête en SQL consituées de plusieurs étapes qu'on peut formaliser à travers des vues.
- Cette optimisation est totalement mécanique. Toutefois, plusieurs résultats sont possibles.

L'optimisation physique

- Elle prend en compte les index, les statistiques (taille des tables et sélectivité des restrictions), la gestion de la mémoire cache et les principes algorithmiques classiques de l'accès au données pour décomposer la requêtes en étapes dont les traitements relèvent de la programmation impérative classique (boucles et tests du langage C).
- L'organisation physique consiste donc à réorganiser les étapes des différentes opérations relationnelles en étapes de programmation impérative.
- Cette optimisation n'est pas totalement mécanique.

Optimisation et heuristique

- **Les optimisation sémantique et syntaxique** sont des étapes algébriques et logiques qui relèvent d'une méthode déductive.
- **L'optimisation physique** et de **choix du plan d'exécution** relèvent d'une méthode heuristique : méthode qui procède par hypothèses provisoires et par évaluations successives.

L'optimisation physique est liée à l'implémentation physique de la BD, mais aussi aux algorithmes de calcul qui sont utilisés. Certains algorithmes sont plus efficaces que d'autres. La qualité de ces algorithmes est un des atouts du SGBD (comme la qualité des algorithmes de recherche est un des atouts des moteurs de recherche).

Optimisation physique et notion de METABASE

L'optimisation physique prend en compte les statistiques de la BD, c'est-à-dire la taille des tables et la sélectivité des restrictions (le nombre de tuples pour une valeur donnée d'un attribut).

Ces informations sont des données sur les données : d'où la notion de **METABASE**.

Le SGBD gère automatiquement la mise à jour de la métabase à l'occasion des requêtes qui lui sont envoyées. Toutefois, la mesure des sélectivités n'est pas faisable à chaque requête sous peine de ralentir son exécution et donc de ne pas atteindre l'objectif d'optimisation. A noter toutefois que, statistiquement, l'ensemble des valeurs possibles pour un attribut est le plus souvent relativement fixe.

Cette mise à jour peut aussi être forcée par l'administrateur de la BD. Des scripts peuvent être écrits pour permettre une mise à jour automatique à heure fixe.

Les données de la métabase participent au choix du plan d'exécution en permettant de faire un calcul de coût le plus réaliste possible.

Plan d'exécution

Un plan d'exécution d'une requête est une suite possible d'opérations de bas niveau (dans un langage cible) qui permettent de réaliser cette requête.

Une requête peut donner lieu à plusieurs plans d'exécution.

Bilan de l'optimisation : le plan d'exécution choisi et calcul de coût

Le choix du plan d'exécution optimal se fera par un calcul de coût, c'est-à-dire de la performance.

Le calcul de coût consiste à calculer le temps de traitement maximum des opérations élémentaires et de leur succession.

Dans le calcul de coût (donc dans la performance de la requête), interviennent :

- Le nombre d'entrée-sortie (accès disque)
- Le temps de calcul des opérations élémentaires
- La taille des buffers requis

Optimisation et méthodes (algorithmes) d'accès aux données

Le critère principal du calcul de coût concerne l'accès aux données.

Il y a deux grands types de méthodes d'accès aux données.

les méthodes par indexation

On retrouve la méthode par indexation dans les SGBD-R. Elle utilile le principe alogorithique de la recherche dichotomique.

les méthodes par hachage

Les méthodes par hachage consistent à utiliser une fonction de calcul qui, appliquée à la clé, détermine l'adresse relative d'un enregistrement. On ne les abordera pas dans ce cours.

La question de la performance : l'évolution des SGBD en terme de rapidité

Capacité de traitement

- **Années 70 :** quelques requêtes par seconde
- **Aujourd'hui :** plusieurs milliers de transactions par seconde

4 causes à l'amélioration des performances des SGBD

- Augmentation de la vitesse du processeur
- Amélioration de la production des plans d'exécution et le leur choix
- Optimisation des méthodes d'accès aux données (les algorithmes de calcul)
- Utilisation de mémoire cache dans les méthodes d'accès aux données

Temps d'entrée-sortie : 10^{-2} seconde

A noter que les temps d'entrée-sortie disque restent à peu près constant : de l'ordre de la **dizaine de millisecondes,** d'où l'intérêt de les limiter par l'optimisation.

OPTIMISATION SYNTAXIQUE

L'optimisation logique met en oeuvre les **arbres algébriques** et les **règles de transformations** qui leur sont associées.

Une fois ces notions présentées, on pourra proposer un **algorithme d'optimisation**.

1. L'arbre algébrique

Exemple traité

On travaille sur l'exemple suivant : (Gardarin, p. 303)

Schéma de la BD

Buveurs (**NB**, Nom, Prénom, Type)

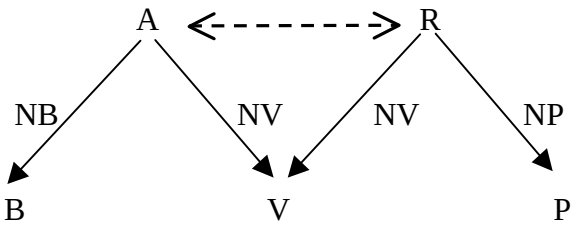
Vins (**NV**, Cru, Millésime, Degré)

Producteurs (**NP**, Nom, Région)

Abuser (**#NB, Date**, Quantité, #NV)

Produire (**#NP, #NV**)

Graphe des tables

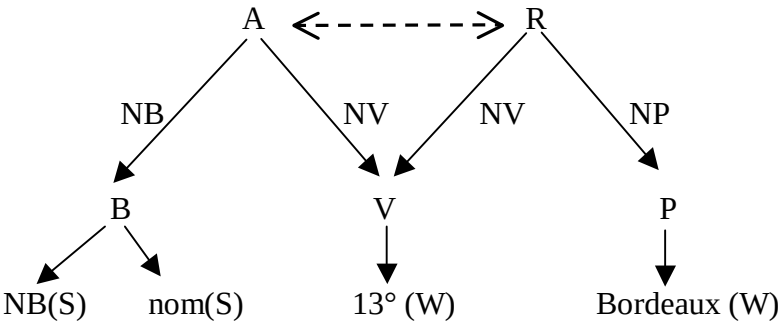


Rappelons que la flèche à double sens en pointillé symbolise une jointure artificielle.

Question traitée

On cherche tous les buveurs ayant bu un Bordeaux de degré supérieur ou égal à 13.

Graphe de la question



Les attributs « S » sont les attributs pour le Select, donc les attributs en sortie de la question.

Les attributs « W » sont les attributs pour le Where, donc les attributs en entrée de la question.

Clé primaire du graphe de la question

La clé primaire du graphe d'une question (hors distinct) est constituée de la concaténation des clés primaires des tables racines du graphe : #NB, Date pour A et #P, #NV pour P.

La clé primaire est donc : #NB, Date, #P, #NV

Réponse SQL

```
Select distinct B.NB, B.Nom
From Buveurs B, Abuser A, Vins V, Produire R, Producteurs P
Where B.NB = A.NB And A.NV =  V.NV
And P.NP = R.NP And R.NV = V.NV
And P.Region = "Bordelais"
And V.Degre >=13;
```

Clé primaire de la question

La question produit des buveurs : la clé primaire sera donc #NB

Etant donné que ce n'est pas la clé primaire du graphe de la question, il faut mettre un distinct pour éliminer les doublons.

Rappel de vocabulaire

Une restriction spécifique est une restriction mono-table.

Une restriction de jointure est une restriction qui met deux tables en jeu.

Une restriction d'agrégat est une restriction qui se fait sur la table résultant de l'agrégat.

Une jointure c'est un produit cartésien associé à une restriction de jointure.

Une jointure naturelle c'est une jointure avec une restriction de jointure naturelle, c'est-à-dire une restriction mettant en jeu une clé étrangère et la clé primaire correspondante.

Table maître et table jointe : en cas de jointure naturelle, on distingue entre la table maître, celle dont la clé étrangère intervient dans la jointure et dont la clé primaire sera la clé primaire de la table résultat, et la table jointe, celle dont la clé primaire intervient dans la jointure.

Une jointure artificielle c'est une jointure avec une restriction de jointure artificielle, c'est-à-dire une restriction de jointure qui n'est pas une restriction de jointure naturelle.

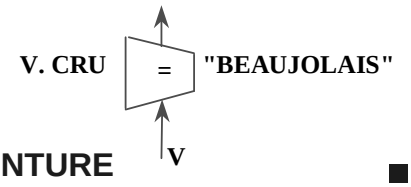
Représentation textuelle des opérateurs de l'algèbre relationnelle

On reprend le formalisme textuel de l'algèbre relationnelle

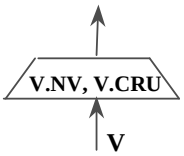
Opération	Formalisme
Projection	P (table ; liste d'attributs)
Restriction	R (table ; liste de restrictions)
Produit cartésien	PC (table1, table 2)
Jointure naturelle	JN (table maître, table jointe ; TM.NTJ=TJ.NTJ)
Jointure artificielle	= PC + Restriction
Union	Union (table1, table2)
Différence	Diff (table1, table2)
Intersection	Inter (table1, table2)
Tri	Tri (table; liste d'attributs)
Agrégat	Agreg (table ; attributs de regroupement ; attributs de statistique)
Fonction de groupe	FG (table ; attributs de statistique)

Représentation graphique des opérateurs de l'algèbre relationnelle dans les arbres algébriques

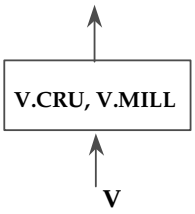
■ **RESTRICTION**



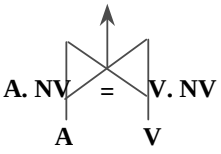
■ **PROJECTION**



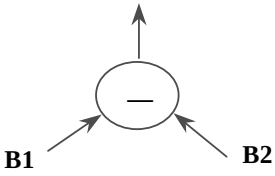
■ **TRI**



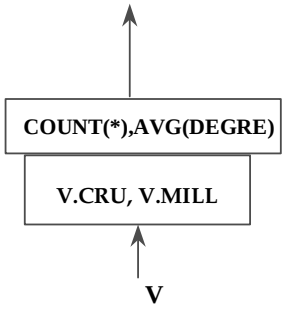
■ **JOINTURE**



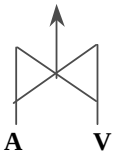
■ **DIFFERENCE**



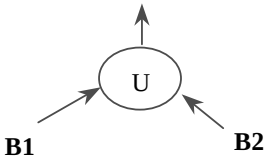
■ **AGREGAT**



■ **PRODUIT CARTESIEN**



■ **UNION**



Cas du distinct

Il n'y a pas de distinct en algèbre relationnelle car par définition une table ne contient pas de doublon et toute opération de l'algèbre relationnelle produit des tables sans doublons.

Arbre algébrique

Présentation

Un arbre algébrique est la représentation d'une requête SQL sous la forme d'un arbre.

Arbre algébrique = arbre relationnel = arbre de traitement

Les feuilles de l'arbre représentent les tables de départ.

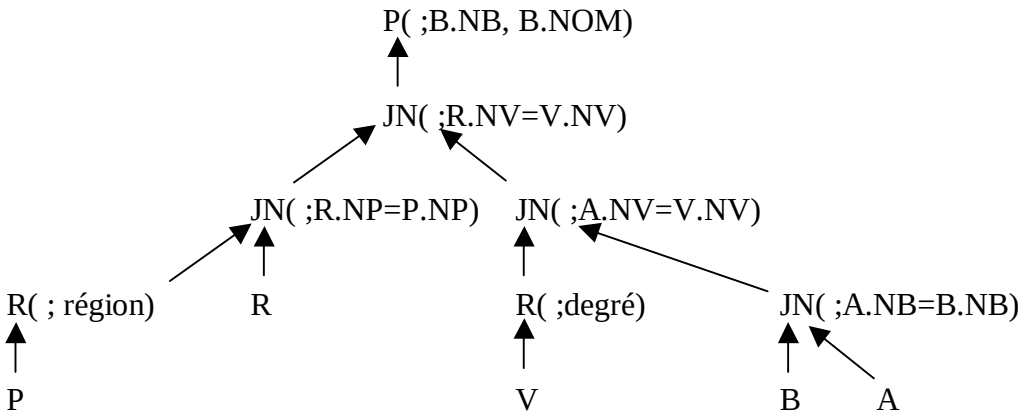
La racine de l'arbre représente la table résultat.

Tous les autres nœuds de l'arbre sont des opérateurs de l'algèbre relationnelle.

Un arbre algébrique correspond à une décomposition de la requête en opérateurs élémentaires avec introduction de tables intermédiaires : c'est donc l'équivalent d'une réécriture de la requête avec des vues.

Arbre algébrique de l'exemple traité

Tous les buveurs ayant bu un Bordeaux de degré supérieur ou égal à 13



Notion d'arbres algébriques équivalent

Notion de requêtes équivalentes

Pour une même question, il peut exister plusieurs requêtes SQL équivalentes à la question.

Deux requêtes sont équivalentes quand elles donnent toujours la même réponse quel que soit le jeu de données auquel elles s'appliquent.

- ***Exemple 1 de requêtes équivalentes :***

Select distinct NB, nom	⇔	Select NB, nom
from Buveurs		from Buveurs

Ces deux requêtes sont équivalentes car la clé primaire du graphe de la question et celle de la question sont identiques.

- ***Exemple 2 de requêtes équivalentes:***

Select distinct NB, nom	**NOT** ⇔	Select NB, nom
from Buveurs, Abus		from Buveurs, Abus
where B.NB = A.NB		where B.NB = A.NB

Les deux requêtes ne sont pas équivalentes car la clé primaire du graphe de la question n'est pas celle de la question.

Notion d'arbres algébriques équivalent

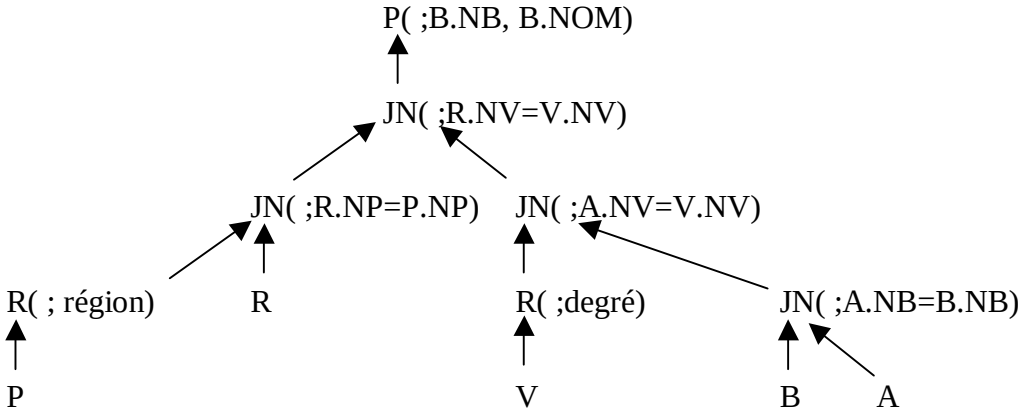
Pour une même requête SQL, on peut construire plusieurs arbres algébriques équivalents.

On distingue entre **arbre linéaire** et **arbre ramifié**.

On distingue aussi entre **arbre optimisé** et **arbre non optimisé**.

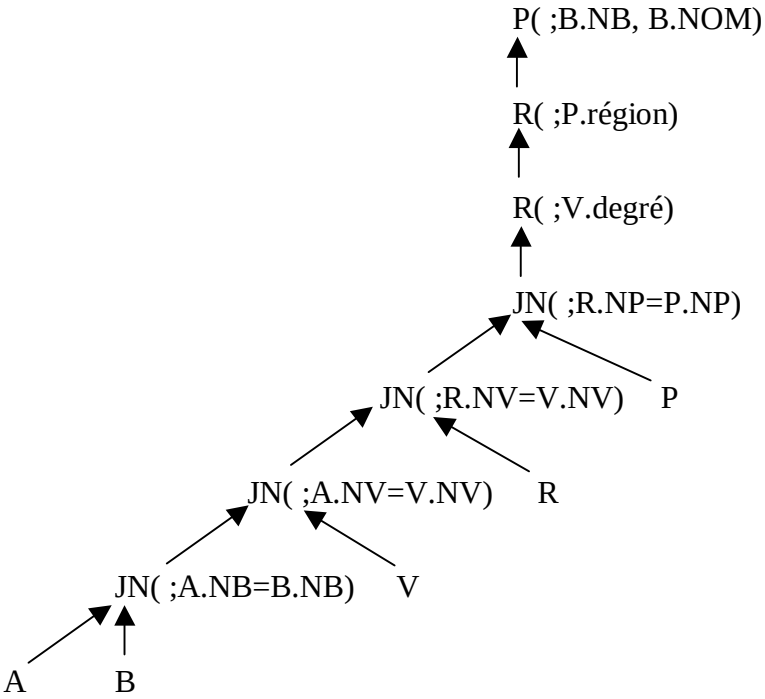
Arbre ramifié

C'est un arbre dont au moins une jointure a deux jointures en entrée :



Arbre linéaire

C'est un arbre dont chaque jointure a au moins une entrée sans jointure :



Nombre théorique d'arbres équivalents pour une requête

En ne s'intéressant qu'aux jointures, on peut calculer le nombre théorique d'arbres équivalents pour une requête en fonction du nombre de tables en jeu.

Avec 2 tables, on peut fabriquer 1 arbre.

Avec 3 tables, on peut fabriquer 3 arbres.

Etc.

La formule générale est donnée par la définition d'une suite :

NA(2) =1
NA(T) = NA(T-1) * (2T-3)

Ce qui nous donne : (2 ;1), (3 ;3), (4 ;15), (5 ; 105), (6 ; 945), (7 ; 10 395), etc.

On arrive donc très vite à un nombre d'arbres équivalents très important. Les algorithmes d'optimisation devront donc aussi être capables de choisir le meilleur.

2. Théorie sur les règles de transformation des arbres algébriques

Il existe plusieurs règles transformation des arbres qui vont permettent de théoriser l'optimisation de l'arbre algébrique d'une requête.

Ces règles proposent des équivalences entre les formulations.

Règles de base sur les produits cartésiens et les jointures

Règle 1 : Commutativité des produits cartésiens et des jointures

JN(A,B) ⇔ JN(B,A)

PC(A,B) ⇔ PC(B,A)

Règle 2 : Associativité des produits cartésiens et des jointures

JN(JN(A,B), C) ⇔ JN (A, JN(B,C))

PC(PC(A,B), C) ⇔ PC (A, PC(B,C))

Règles de base sur les restrictions

Formalisme

→ signifie : suivi de

A → B est équivalent à :

Règle 3 : Fusion et dissociation des restrictions

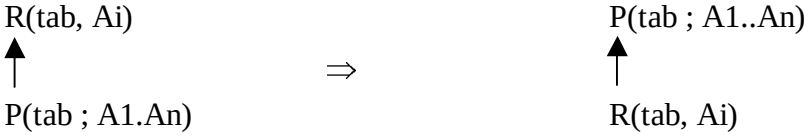
R(table ; restiction1, restriction2) ⇔ R(table ; restiction1) → R(table ; restriction2)

Règle 4 : Commutativité des restrictions

R(table ; restiction1, restriction2) ⇔ R(table ; restiction2, restriction1)

Règle 5 : Commutativité sous condition des restrictions et des projections

On peut toujours faire la restriction avant la projection :



On peut faire la restriction après la projection sous certaines conditions :

⇐

Si Ai ∈ A1..An

Règle 6 : Commutativité sous condition des restrictions et des produits cartésiens (et aussi des jointures)

On peut toujours faire la restriction après le produit cartésien ou la jointure :

On peut faire la restriction avant le produit cartésien ou la jointure sous certaines conditions :

⇐

Si Ai ∈ A1..An

Règle 7 : Commutativité des restrictions avec les opérations ensemblistes

R(tab1;Ai..Ak) et R(tab2;Ai..Ak) → Union (tab1, tab2) ⇔ Union (tab1, tab2) → R(;Ai..Ak)

R(tab1;Ai..Ak) et R(tab2;Ai..Ak) → Inter (tab1, tab2) ⇔ Inter (tab1, tab2) → R(;Ai..Ak)

R(tab1;Ai..Ak) et R(tab2;Ai..Ak) → Diff (tab1, tab2) ⇔ Diff (tab1, tab2) → R(;Ai..Ak)

Règles de base sur les projections

Règle 8 : Fusion et dissociation des projections

P(table ; Att1, Att2) ⇔ P(table ; Att1) → P(table ; Att2)

Règle 9 : Commutativité des projections

P(table ; Att1, Att2) ⇔ P(table ; Att2, Att1)

Règle 10 : Commutativité sous condition des projections et des produits cartésiens

On peut toujours faire la projection après le produit cartésien

P(tab, Ai) → PC(table1(A1…An), table2) ⇒ PC(table1(A1…An), table2) → P(tab, Ai)

On peut faire la projection avant le produit cartésien sous certaines conditions:

P(tab, Ai) → PC(table1(A1…An), table2) ⇐ PC(table1(A1…An), table2) → P(tab, Ai)

Si Ai ∈ A1..An

Règle 11 : Commutativité sous condition des projections et des jointures

On peut toujours faire la projection après la jointure

P(tab1, Ai..Ak) → PC(tab1(A1..An), tab2) ⇒ PC(tab1(A1..An), tab2) → P(;Ai..Ak)

On peut faire la projection avant la jointure sous certaines conditions:

P(tab1, Ai..Ak) → PC(tab1(A1..An), tab2) ⇐ PC(tab1(A1..An), table2) → P(; Ai..Ak)

Si Ai..Ak ∈ A1..An et si l'attribut de jointure ∈ Ai..Ak

Règle 12 : Commutativité des projections avec les opérations ensemblistes

P(tab1;Ai..Ak) et P(tab2;Ai..Ak) → Union (tab1, tab2) ⇔ Union (tab1, tab2) → P(;Ai..Ak)

P(tab1;Ai..Ak) et P(tab2;Ai..Ak) → Inter (tab1, tab2) ⇔ Inter (tab1, tab2) → P(;Ai..Ak)

P(tab1;Ai..Ak) et P(tab2;Ai..Ak) → Diff (tab1, tab2) ⇔ Diff (tab1, tab2) → P(;Ai..Ak)

3. Algorithme d'optimisation syntaxique et arbres algébriques optimisés

Algorithme d'optimisation syntaxique

Principe intuitif de l'optimisation intuitive

Le principe général de l'optimisation repose sur le constat suivant :

- Les opérations unaires produisent des tables plus petites que la table d'origine.
- Les opérations binaires produisent des tables plus grandes que la table d'origine. Autrement dit, ce sont les produits cartésiens des jointures qui accroissent la taille des tables intermédiaires.

On va donc :

Supprimer un maximum de lignes et de colonnes avant de faire les jointures.

Faire les jointures naturelles avant les jointures artificielles, elles-mêmes encore avant les produits cartésiens.

Etapes de l'optimisation intuitive

L'optimisation intuitive se résume à :

1. Faire toutes les restrictions spécifiques pour limiter le nombre de tuples.
2. Faire toutes les projections mono-tables possibles pour limiter le nombres d'attributs.
3. Faire les jointures et après chaque jointure, les projections possibles.
4. Finir la projection
5. Faire les distinct, les tris, les group by, les fonctions de groupe.

L'optimisation d'après les règles de transformation

- En utilisant les règles de transformations précédentes, on arrive à l'algorithme suivant :
-

1. Règle 3 : on sépare les restrictions comportant plusieurs prédicats.
2. Règles 4, 5, 6 et 7 : on descend les restrictions le plus bas possible.
3. Règles 3 : on regroupe les restrictions successives
4. Règle 8 : on sépare les projections comportant plusieurs prédicats.
5. Règle 9, 10, 11 et 12 : on descend les projections le plus bas possibles
6. Règle 8 : on regroupe les projections successives

Arbres algébriques optimisés

On peut maintenant optimiser les deux arbres algébriques précédemment présentés.

Rappel du modèle :

Buveurs (**NB**, Nom, Prénom, Type)

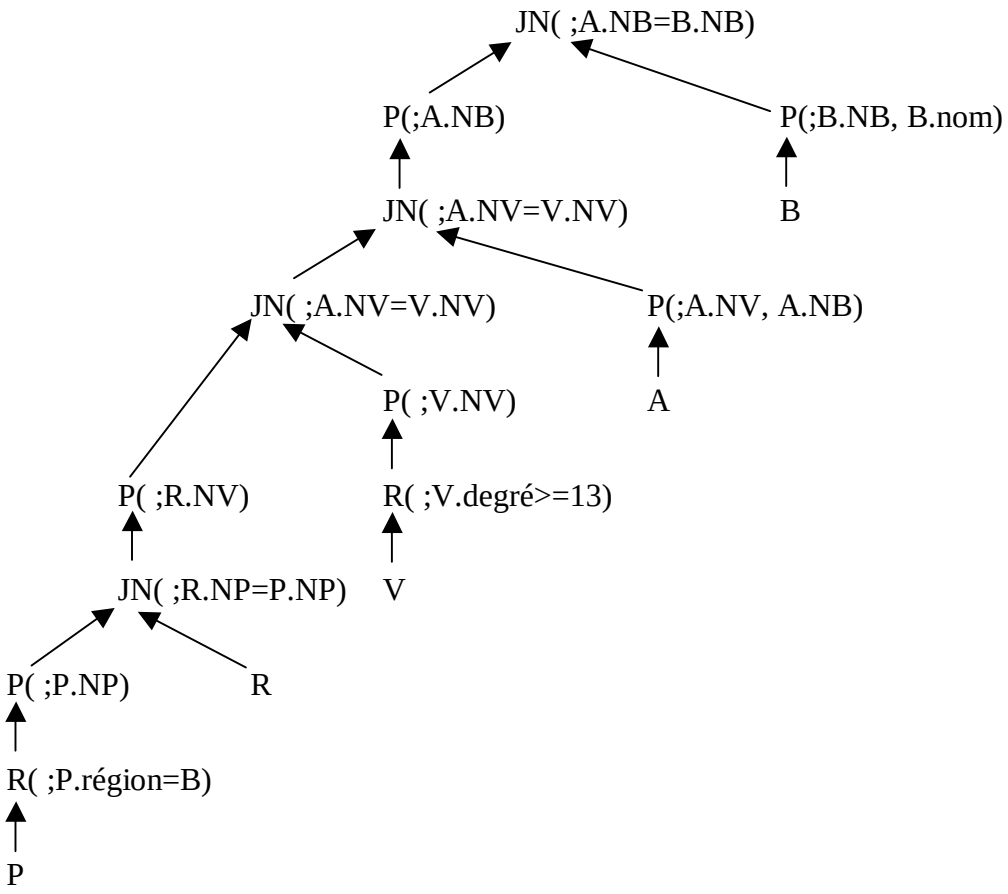
Vins (**NV**, Cru, Millésime, Degré)

Producteurs (**NP**, Nom, Région)

Abuser (**#NB, Date**, Quantité, #NV)

Produire (**#NP, #NV**)

Arbre linéaire optimisé

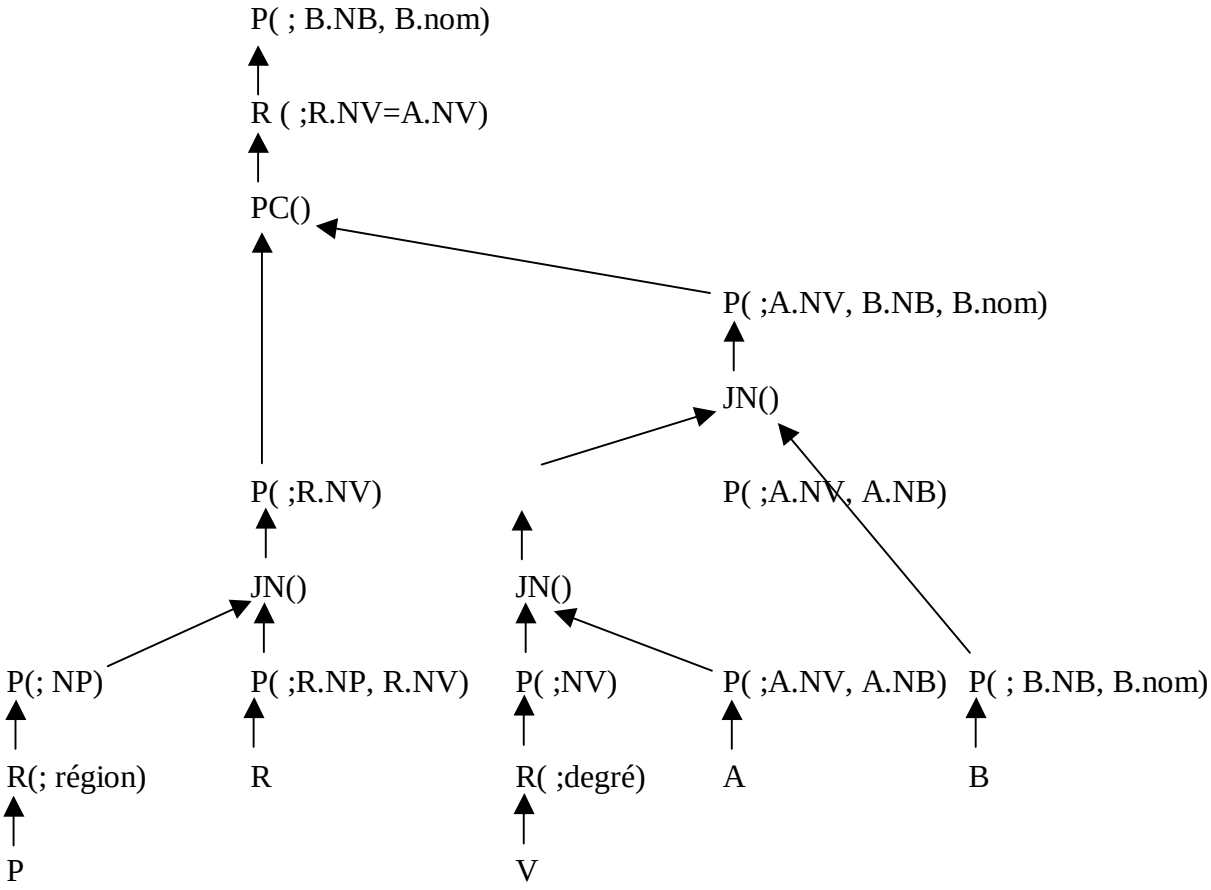


Traduction SQL

```
Create or replace view v1 as Select * from producteurs where region = “Bordelais”;
Create or replace view v2 as Select np from v1;
Create or replace view v3 as Select * from produire join v2 using (np);
Create or replace view v4 as Select nv from v3;
Create or replace view v5 as Select * from vins where degre >=13;
Create or replace view v6 as Select nv from v5;
Create or replace view v7 as Select * from v4 join v6 using (nv);
Create or replace view v8 as Select nv, nb from abuser;
Create or replace view v9 as Select * from v7 join v8 using (nv);
Create or replace view v10 as Select nb from v9;
Create or replace view v11 as Select nb, nom from buveurs;
Create or replace view v12 as Select * from v10 join v11 using (nb);
Select * from v12;
```

Ce dernier select produit le même résultat que le select de base.

Arbre ramifié optimisé



CALCUL DE COUT ET OPTIMISATION PHYSIQUE

1. Première approche du calcul du coût : approche SQL

Quantité de données

Rappel du modèle

Buveurs (**NB**, Nom, Prénom, Type)
Vins (**NV**, Cru, Millésime, Degré)
Producteurs (**NP**, Nom, Région)
Abuser (**#NB, Date**, Quantité, #NV)
Produire (**#NP, #NV**)

Taille des tables

La taille des tables prend en compte le nombre de tuples et le nombre d'attributs. Pour être précis, il faudrait prendre en compte la taille des attributs.

- 100 buveurs : **B = (100 ; 4)**
- 50 vins : **V = (50 ; 4)**
- 20 producteurs : **P = (20 ; 3)**
- 250 abuser : **A = (250 ; 4)**
- 75 produire : **R = (75 ; 2)**

Sélectivité

La sélectivité, c'est le pourcentage de tuples qui vérifient une condition.
La sélectivité est un élément de la métabase.

On ajoute les sélectivités suivantes :

- Sélectivité(Producteurs ; Région = "Bordeaux") = 25%
- Sélectivité(Vins ; Degré >13) =20%

Principes du calcul

Tables d'origines

On note le nombre de tuples et d'attributs dans chaque tables après chaque opération.

En cas de produit cartésien

Le nombre de tuples de la table résultat est égal à la multiplication des nombres de tuples des 2 tables du produit cartésien.

Le nombre d'attributs est égal à la somme des nombres d'attributs des deux tables.

En cas de jointure artificielle

On ne peut rien dire sur le nombre de tuples et on en revient à un produit cartésien.

En cas de jointure naturelle

On retrouve exactement le nombre de tuples de la table maître si la table jointe n'a pas subi de restriction, sinon on retrouve au maximum le nombre de tuples de la tables maître.

Le nombre d'attributs est égal à la somme des nombres d'attributs des deux tables –1 : on considère qu'un des deux attribut de liaison est automatiquement supprimé.

En cas de restriction

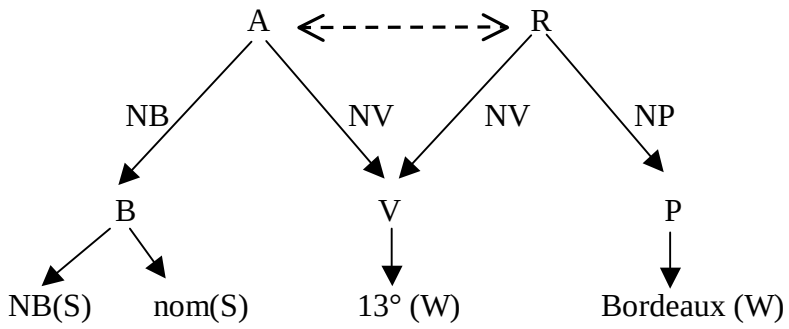
Le coefficient de sélectivité d'applique directement si la restriction s'applique immédiatement à la table, sinon il est inutilisable.

Exemple

Question traitée

On cherche tous les buveurs ayant bu un Bordeaux de degré supérieur ou égal à 13.

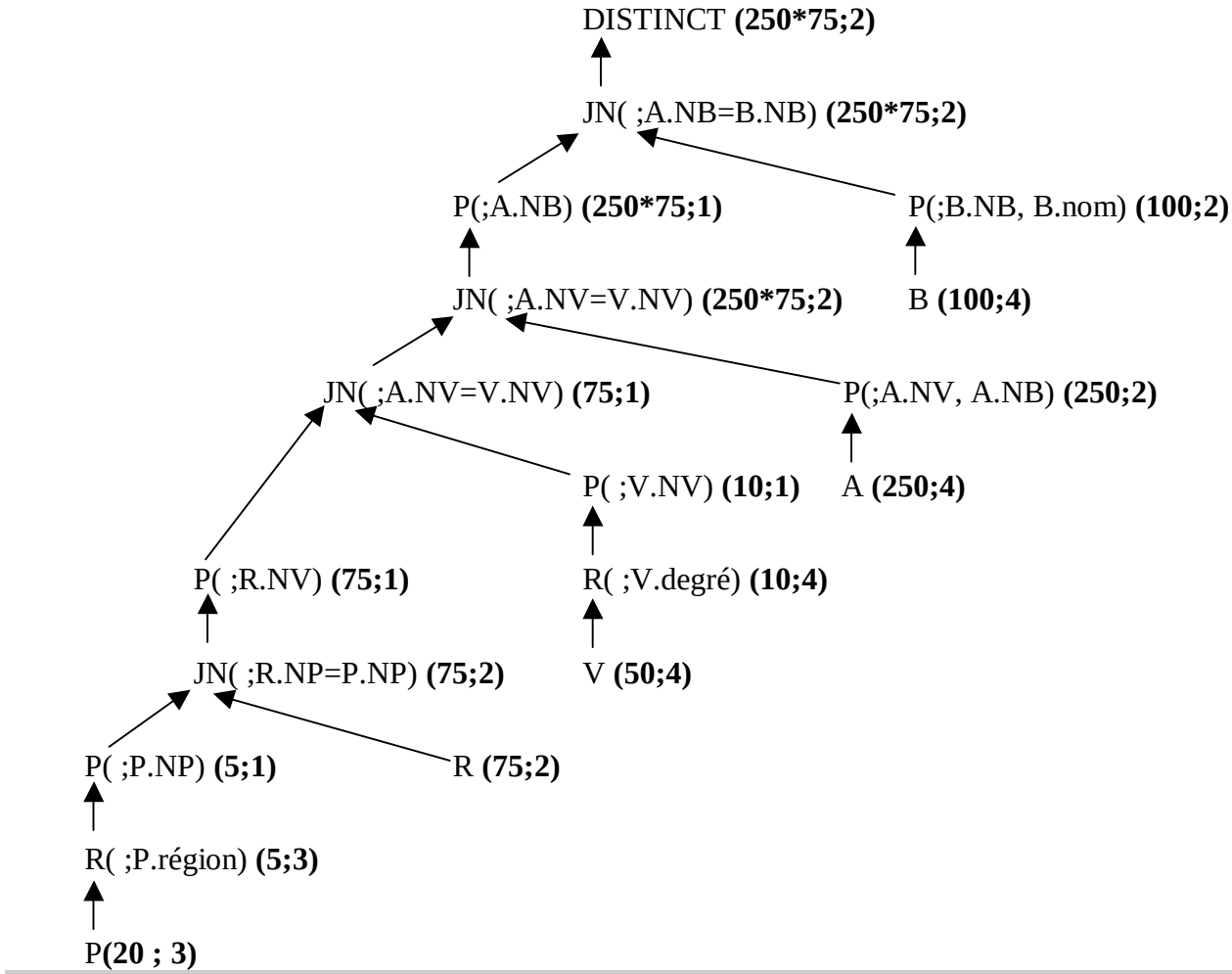
Graphe de la question



Réponse SQL

Select distinct B.NB, B.Nom
From Buveurs B, Abuser A, Vins V, Produire R, Producteurs P
Where B.NB = A.NB And A.NV = V.NV
And P.NP = R.NP And R.NV = V.NV
And P.Region = “Bordelais”

And V.Degre >=13;Arbre linéaire optimisé avec calcul de coût



2. Optimisation phyisique et calcul de coût indexé

Principe du passage du SQL au langage impératif

Le passage du SQL au langage impératif va essentiellement **mixer** l'analyse de **l'arbre algébrique** et les possibilités offertes par **les index** et l'algorithmique des index.

Pour cela, on va définir la **jointure indexée** qui est une opération d'algorithmique impérative.

La jointure indexée

Présentation

- opération d'algorithmique impérative
- non commutative
- Principe : on part d'une première table que l'on parcourt entièrement et, pour chaque tuple, on fait la jointure avec la deuxième table en utilisant un index de cette deuxième table ce qui permet un accès direct.

Exemple

On veut tous les abus avec le nom des buveurs.

- ***Réponse SQL***

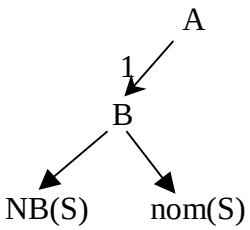
```
Select distinct a.nb, b.nom
from abuser a, buveurs b
where a.nb=b.nb ;
```

On fait une jointure indexé entre ABUSER et les BUVEURS qui consistera à récupérer, pour chaque tuple de ABUSER, le tuple de BUVEURS correspondant.

Le lien se faisant par NB qui est un attribut indexé de BUVEURS, la recherche sera une recherche dichotomique. Le nombre d'accès disques pour récupérer les buveurs des abus sera celui du nombre de tuples dans la table ABUSER.

On a alors une jointure indexée unique.

- ***Graphe des jointures indexés***



La **flèche avec un 1** signifie qu'on a une **jointure indexée unique**.

Distinction entre jointure indexée unique (EQ_REF) et jointure indexé multiple (REF)

Exemple

Dans l'exemple précédent, on a une **jointure indexée unique** (**EQ_REF** dans le vocabulaire de l'EXPLAIN MySQL) car l'index utilisé est un **index unique** (index de la clé primaire des BUVEURS : NB.

Une **jointure indexée multiple** (**REF** dans le vocabulaire de l'EXPLAIN MySQL) est une jointure indexée qui utilise un **index multiple** (non unique).

Exemple

On veut maintenant tous les abus leur date et avec le nom des buveurs et le cru du vin.

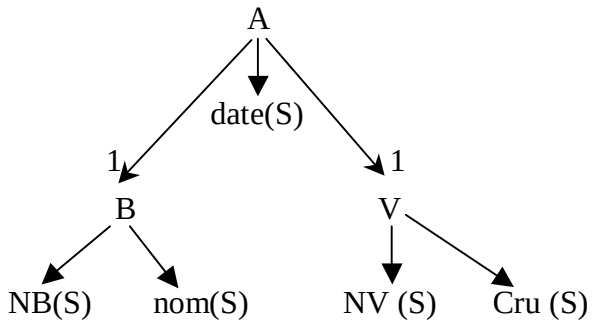
Réponse SQL

```
Select a.nb, b.nom, a .date, v.nv, v.cru
from abuser a, vins v, buveurs b
where a.nv=v.nv and a.nb=b.nb ;
```

Première solution

On peut commencer par faire une jointure indexée unique entre ABUSER et BUVEURS puis une autre jointure indexée unique entre ABUSER et VINS.

- ***Graphe des jointures indexés***



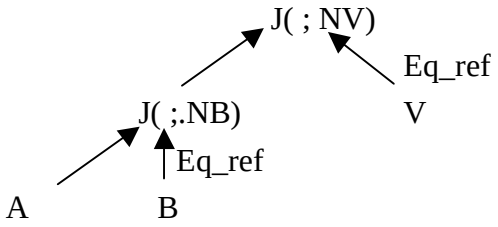
La **flèche avec un 1** signifie qu'on a une **jointure indexée unique**.

- ***Arbre algébrique indexé***

Ce type d'arbre est **toujours linéaire** et utilise des **jointures indexées uniques et multiples**.

Il n'est pas nécessaire de préciser les projections.

Par contre on précisera les restrictions spécifiques s'il y a lieu.



Le caractère non commutatif de la jointure et son type unique ou multiple est précisé sur la branche de la table contenant l'index. EQ_REF pour unique, REF pour multiple.

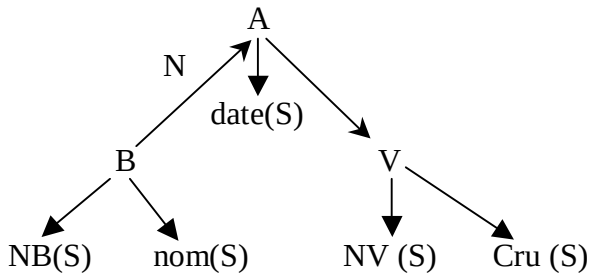
L'arbre se lit ainsi : on part des ABUSER, on y associe les BUVEURS par jointure indexées unique puis les VINS par jointure indexées unique.

Deuxième solution

On peut aussi commencer par une jointure indexée multiple entre BUVEURS et ABUSER (car le NB de ABUSER est indexé en tant que clé primaire) qui sera suivi par une jointure indexée unique entre ABUSER et VINS. A noter qu'on ne pourrait pas faire une jointure indexée multiple entre BUVEURS et VINS car le NV de ABUSER n'est pas indexé.

La jointure indexée multiple consiste à partir de la table des BUVEURS, puis à récupérer, pour chaque tuple de BUVEURS, les tuples de ABUSER correspondant.

- ***Graphe des jointures indexés***



La **flèche avec un 1** signifie qu'on a une **jointure indexée unique**.

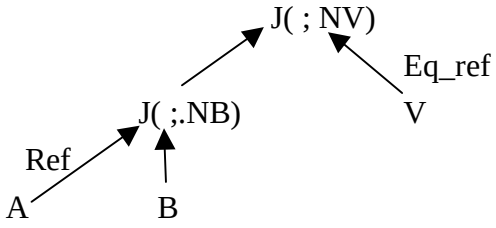
La **flèche avec un N** signifie qu'on a une **jointure indexée multiple**.

- ***Arbre algébrique indexé***

Ce type d'arbre est **toujours linéaire** et utilise des **jointures indexées uniques et multiples**.

Il n'est pas nécessaire de préciser les projections.

Par contre on précisera les restrictions spécifiques s'il y a lieu.



Le caractère non commutatif de la jointure et son type unique ou multiple est précisé sur la branche de la table contenant l'index. EQ_REF pour unique, REF pour multiple.

L'arbre se lit ainsi : on part des BUVEURS, on y associe les ASSOCIER par jointure indexée multiple puis les VINS par jointure indexée unique.

Pourquoi choisir une solution plutôt qu'une autre ?

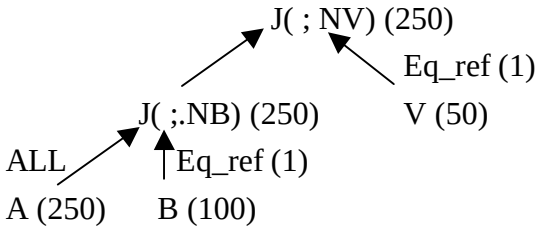
C'est l'optimiseur qui tranche, en fonction des index, des statistiques de la métabase et de son heuristique.

Le principe général consiste à limiter le nombre d'accès disques, soit, en première approximation, le nombre de tuples parcourus.

Calcul de coût et arbre algébrique indexé

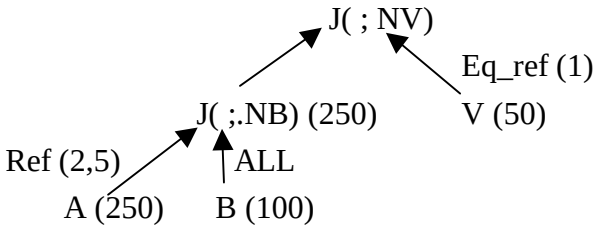
On va inscrire dans l'arbre algébrique indexé le nombre de tuples récupés dans chaque table.

Première solution



Le nombre d'accès diques, en première approximation, sera: 250 + 250*1 + 250*1 = 750
(250 de A + 250 de A * 1 de B + 250 de J(A,B) * 1 de V)

Deuxième solution



Le 2,5 correspond au nombre moyen de tuples de ABUSER à récupérer à chaque recherche indexée à partir d'un tuple de BUVEURS : c'est 250/100.

Le nombre d'accès diques, en première approximation, sera: 100 + 100*2,5 + 250*1 = 600
(100 de B + 100 de B * 2,5 de A + 250 de J(A,B) * 1 de V)

Cette solution est donc meilleure que la précédente : ce sera celle proposée par l'EXPLAIN.

Les restrictions dans les arbres algébriques indexés

Exemple

On veut maintenant tous les abus de quantité = 4 avec leur date et avec le nom des buveurs et le cru du vin.

On pose une sélectivité de 5% pour la restriction quantité=4

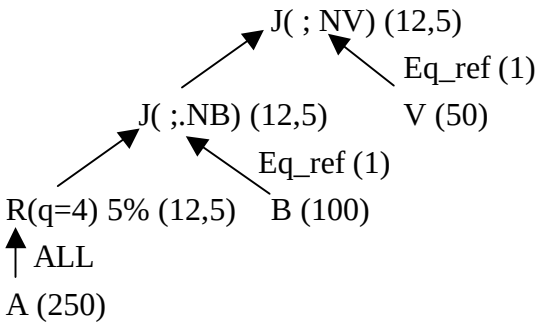
- ***Réponse SQL***

Select a.nb, b.nom, a .date, v.nv, v.cru from abuser a, vins v, buveurs b where a.nv=v.nv and a.nb=b.nb and quantite = 4;

Première solution

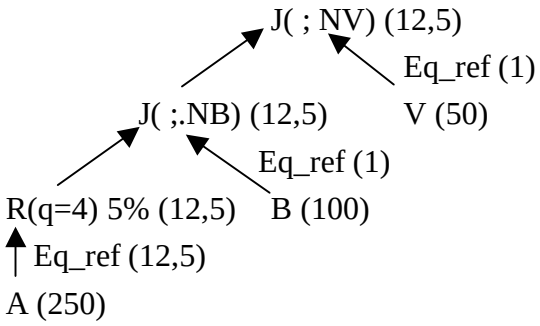
On commence par faire une restriction sur ABUSER puis une jointure indexée unique entre ABUSER et BUVEURS puis une autre jointure indexée unique entre ABUSER et VINS.

- ***Arbre algébrique indexé et calcul de coût***



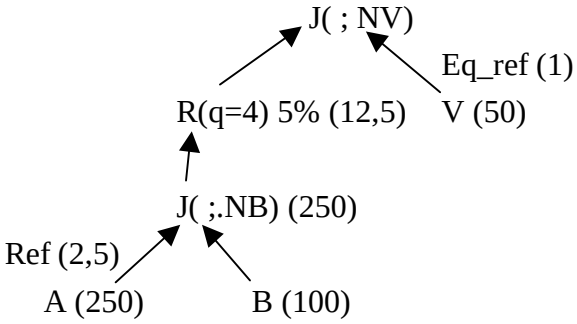
Le nombre d'accès diques, en première approximation, sera: 250 + 12,5*1 + 12,5*1 = 300
(250 de A + 12,5 de A restreint * 1 de B + 12,5 de J(A,B) * 1 de V)

Les 250 de A pourrait être ramené à 12,5 si l'attribut quantité était indexé : on obtiendrait alors 37,5 au calcul de coût.



Deuxième solution

On commence par une jointure indexée multiple entre BUVEURS et ABUSER (car le NB de ABUSER est indexé en tant que clé primaire) qui sera suivi par une jointure indexée unique entre ABUSER et VINS.



La restriction à 5 % intervient après chaque accès (par la jointure indexée multiple) aux tuples de ABUSER correspondant à un tuple de BUVEURS.

Le 2,5 correspond au nombre moyen de tuples de ABUSER à récupérer à chaque recherche indexée à partir d'un tuple de BUVEURS : c'est 250/100.

Le nombre d'accès diques, en première approximation, sera: 100 + 100*2,5 + 12,5*1 = 362,5

(100 de B + 100 de B * 2,5 de A =250 + 12,5 de J(A,B) * 1 de V)

La restriction n'a pas d'influence sur les accès disques.

Conclusion

Les restrictions sur la table d'index interviennent après la jointure indexée.

Les deux solutions se valent (300 contre 362,5).

L'ajout d'un index rend la première solution nettement plus performante (37, contre 300).

EXERCICES

1. Les employés

EMPLOYES(<u>NE</u>, nom, job, datemb, sal, comm., #ND, *NEchef)

DEPARTEMENTS (<u>ND</u>, nom, ville)

Employés(500 ; 8)

Départements (5 ; 3)

S (départements ; nom= « Comptabilité ») = 20%

Requête : Liste des employés par ordre alphabétique du département comptabilité avec leurs jobs et le nom de leur supérieur hiérarchique

- Faire le graphe des tables
- Faire le graphe de la question
- Répondre à la question en SQL
- Faire un arbre algébrique ramifié.
- Faire un arbre linéaire et optimisé avec calcul de coût SQL
- Traduire cet arbre en SQL avec des vues.
- Faire un arbre linéaire et optimisé avec un calcul de coût indexé
- Quelle optimisation peut-on envisager ?

2. La bibliothèque

ADHERENTS (NA, nom, prenom, adr, CP, ville, tel)

OEUVRES (NO, titre, auteur)

LIVRES (NL, editeur, #NO)

EMPRUNTER(#NL, datemp, datretmax, datret, #NA)

Les livres sont les exemplaires physiques des livres.

Adhérents(10 000 ; 7)

Oeuvres (1 100 000 ; 3)

Livres (1 200 000 ; 3)

Emprunter (500 000 ; 5)

S (Adhérents ; CP = 75 019) = 10%

S (Emprunter ; dateRet is NULL) = 5%

Requête : tous les auteurs actuellement empruntés par des adhérents du 19ème arrondissement, trié par ordre alphabétique.

- Faire le graphe des tables
- Faire le graphe de la question
- Répondre à la question en SQL
- Faire un arbre algébrique ramifié.
- Faire un arbre linéaire et optimisé avec calcul de coût SQL
- Traduire cet arbre en SQL avec des vues.
- Faire un arbre linéaire et optimisé avec un calcul de coût indexé
- Quelle optimisation peut-on envisager ?

3. Les projets

Schéma de la BD projets simplifié

EMPLOYES (**NE,** nom)

SERVICES (**NS,** Sx, #NE) Un service a un chef de service.

PROJETS (**NP**, Px, #NS) Un projet est affecté à un service et un seul.

PARTICIPER (**#NE, #NP**) Les employés participent à des projets.

Les chefs de services ne participent pas forcément aux projets affectés à leur service.

Employes (1 000 ; 2)

Services (10 ; 3)

Projets (500 ; 3)

Participer (10 000 ; 2)

Tous les projets sur lesquels Jean Dupont a travaillé et dont il a été le chef de service du service responsable du projet

- Faire le graphe des tables
- Faire le graphe de la question
- Répondre à la question en SQL
- Faire un arbre algébrique ramifié.
- Faire un arbre linéaire et optimisé avec calcul de coût
- Traduire cet arbre en SQL avec des vues.
- Faire un arbre linéaire et optimisé avec un calcul de coût indexé
- Quelle optimisation peut-on envisager ?

4. La maison de disques

Schéma de la BD Maison de disques

DISQUES (**ND**, titre, année)

CHANSONS (**NC**, titre, durée, année)

MUSICIENS (**NM**, nom, prénom, nationalité)

JOUER (**#NM, #NC, instrument**)

REGROUPER ((#ND, #NC, piste**)**

Disques (100 000 ; 3)

Chansons (1 000 000 ; 3)

Musiciens (10 000 ; 4)

Jouer (4 000 000 ; 3)

Regrouper (700 000 ; 3)

S (Disques ; année >= 1960) = 80%

Requête : Quels sont les titre des disques auxquels a participé KUTI en tant que saxophoniste depuis 1960.

- Faire le graphe des tables
- Faire le graphe de la question
- Répondre à la question en SQL
- Faire un arbre algébrique ramifié.
- Faire un arbre linéaire et optimisé avec calcul de coût
- Traduire cet arbre en SQL avec des vues.
- Faire un arbre linéaire et optimisé avec un calcul de coût indexé
- Quelle optimisation peut-on envisager ?

5. L'école

Schéma de la BD Ecole

ETUDIANTS (**NET**, nom, prenom, email)

GROUPES (**NGR**, nom, niveau, num, année) : exemple : 37, SIGL, 3, 1, 2008 : SIGL3 groupe 1 de 2008-2009.

EXAMENS (**NEX**, matiere, prof, sujet, dateCréation, niveauPrévu, typePrévu, duréePrévue) : exemple de niveau préu : 3 pour troisième année.

EPREUVES (**NEP**, #NGR, dateHeureDébut, durée, type, salle, #NEX) : #NGR et dateheure forment une clé secondaire. A noter que #NGR et salle aussi.

EVALUER (**#NEP, #NET**, note) Les étudiants sont évalués dans une épreuve: il y a une note. En cas d'absence, ils ne sont pas dans la table.

PARTICIPER (#NET, #NGR) : les étudiants participent à un groupe. Ils peuvent participer à plusieurs groupes.

Fonctionnement du système : on crée un groupe, on enregistre les étudiants et on les affecte dans un groupe (table participer). On crée des examens, puis des épreuves qui font référence à ces examens. L'épreuve est pour un groupe donné. Quand on récupère les copies, on enregistre des évaluations avec des notes.

Etudiants (1 000 ; 4)

Groupe (115 ; 5)

Examens (2 800 ; 8)

Epreuves (4 000 ; 7)

Evaluer (40 000 ; 3)

Participer (39 000 ; 2)

S (Groupe ; Niveau = 3) = 38 %

S (Groupe ; Année = 2007) = 10%

S (Examens ; Matière = BD) = 10%

S (Groupe ; Niveau = 2 et nom = SIGL) = 15%

S (Examens ; Matière = Math) = 10%

(Remarque : on part sur un modèle de 10 ans, 100 élèves par promo, 3 années, 3 groupes (1 classe) en 1ère année, 5 groupes (3 classes) en 2ème et 3ème année, 20 matières par an pour une classe, 2 examens par matières.

Requête 1 : Les notes des élèves de 3ème année pour les épreuves de BD pour l'année scolaire 2007-2008 ?

Le résultat devra être exploitable. Dans les résultats, on précisera la matière, son type et la date de l'examen.

Requête 2 : Quels sont les étudiants absents aux épreuves de math de l'année 2007 en SIGL 2.

Pour répondre à cette question, on a intérêt à se demander qui étaient les étudiants prévus et qui étaient les étudiants présents.

Pour chaque requête :

- Faire le graphe des tables
- Faire le graphe de la question
- Répondre à la question en SQL
- Faire un arbre algébrique ramifié.
- Faire un arbre linéaire et optimisé avec calcul de coût
- Traduire cet arbre en SQL avec des vues.
- Faire un arbre linéaire et optimisé avec un calcul de coût indexé
- Quelle optimisation peut-on envisager ?