Plan : Optimisation de requêtes

1. Introduction.
    1. Intérêt.
    2. Les étapes d'une optimisation.
2. Réécriture de requête.
3. Les informations utiles à l'optimisation.
4. Différents algorithmes de calcul de jointures.
5. Divers.
    1. Traitement de l'opérateur IN.
    2. Recherche de sous expressions communes.
    3. …

1. Introduction 1. Intérêt

Exemple :

- FABRICANT (NFabricant, Ville) 100 fabricants
- VEND (NF, NP) 10000 enregistrements
- PIECE (NPiece, Descriptif)
- 100 fabricants, chaque fabricant vend 100 pièces, la pièce 12 est vendue par 10 fabricants.

Ville où se trouvent les fabricants de la pièce 12 :

```
SELECT DISTINCT ville
    FROM fabricant, vend
    WHERE NFabricant= NF AND NP = 12.
```

Comment le SGBD peut-il calculer effectivement le résultat ?

1. Introduction 1. Intérêt

Plusieurs manières de procéder :

- **A) avec produit cartésien**.

  R1 = Fabricant X Vend ;

  R2 = SELECT (R1, *NFabricant* = *NF*) ;

  R3 = SELECT (R2, *NP* = *12*) ;

  Résultat = PROJECTION (R3,Ville)

- **B) avec produit cartésien.**

  R1 = SELECT(Vend, *NP* = *12*) ;

  R2 = Fabricant X R1 ;

  R3 = SELECT (R2, *NF* = *NFabricant*) ;

  Résultat = PROJECTION (R3,Ville)

- **C) avec une jointure.**

  R1 = SELECT(Vend, *NP* = *12*) ;

  R2 = JOINTURE (Fabricant, R1, *NFabricant* = *NF*) ;

  Résultat = PROJECTION (R2,Ville) ;

1. Introduction 1. Intérêt

Taille (en nombre d'articles) des relations intermédiaires à manipuler et stocker ?

- **a) avec produit cartésien**.
  R1 = Fabricant X Vend ; 100 x 10000 = 10^6
  R2 = SELECT (R1, *NFabricant* = *NF*) ; 10000 = 10
  R3 = SELECT (R2, *NP* = *12*) ; 10
  Résultat = PROJECTION (R3,Ville)
- **b) avec produit cartésien.**
  R1 = SELECT(Vend, *NP* = *12*) ;
  R2 = Fabricant X R1 ; 10 x 100 = 10^3
  R3 = SELECT (R2, *NF* = *NFabricant*) ; 10
  Résultat = PROJECTION (R3,Ville)
- **c) avec une jointure.**
  R1 = SELECT(Vend, *NP* = *12*) ;
  R2 = JOINTURE (Fabricant, R1, *NFabricant* = *NF*) ;
  Résultat = PROJECTION (R2,Ville) ;

Algorithme de calcul de la jointure du c) ?

Nombre de tests ?

- **i) 2 boucles imbriquées.**

Pour chaque ligne *F* de *Fabricant* faire — **100 fois**
    Pour chaque ligne *R* de *R1* faire — **10 fois**
        Si *F.NFabricant = R.NF* alors
            rajouter (*F & R*) dans *R2*.
        FinSi ;
    FinPour ;
FinPour ;

=> 10 x 100 = 1000 tests

Algorithme de calcul de la jointure du c) ?

Nombre de tests ?

- **ii) en utilisant un index sur *NFabricant* (c'est la clé).**

Pour chaque ligne *R* de *R1* faire **10 fois**
    Chercher dans *Fabricant* le(s) article(s) tels que (*NFabricant = R.NF).*
    Pour chacun des articles *Fi* trouvés faire
        Rajouter (*Fi & R*) dans *R2*.
    FinPour ;
FinPour ;

En *O(Log(100))* = *7en base 2 si arbre binaire de recherche ou dichotomie*

=> 10 x log(100) ≈ 70 tests

Algorithme de calcul de la jointure du c) ?

Nombre de tests ?

- **Iii) si index sur *R1.NF* (index sur *Vend.NF* ?)**

Pour chaque ligne *F* de *Fabricant* faire **100 fois**
    Chercher dans *R1* le(s) article(s) tels que (*F.NFabricant = NF)*
    Pour chacun des articles *Ri* trouvés faire
        Rajouter (*F & Ri*) dans *R2*.
    FinPour ;
FinPour ;

En *O(Log(10))* = *4 si arbre binaire de recherche ou dichotomie*

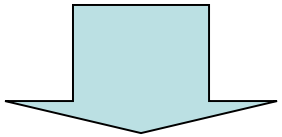
=> 100 x log(10) ≈ 400 tests

1. Introduction 1. Intérêt

100000 articles créés et tests de jointure

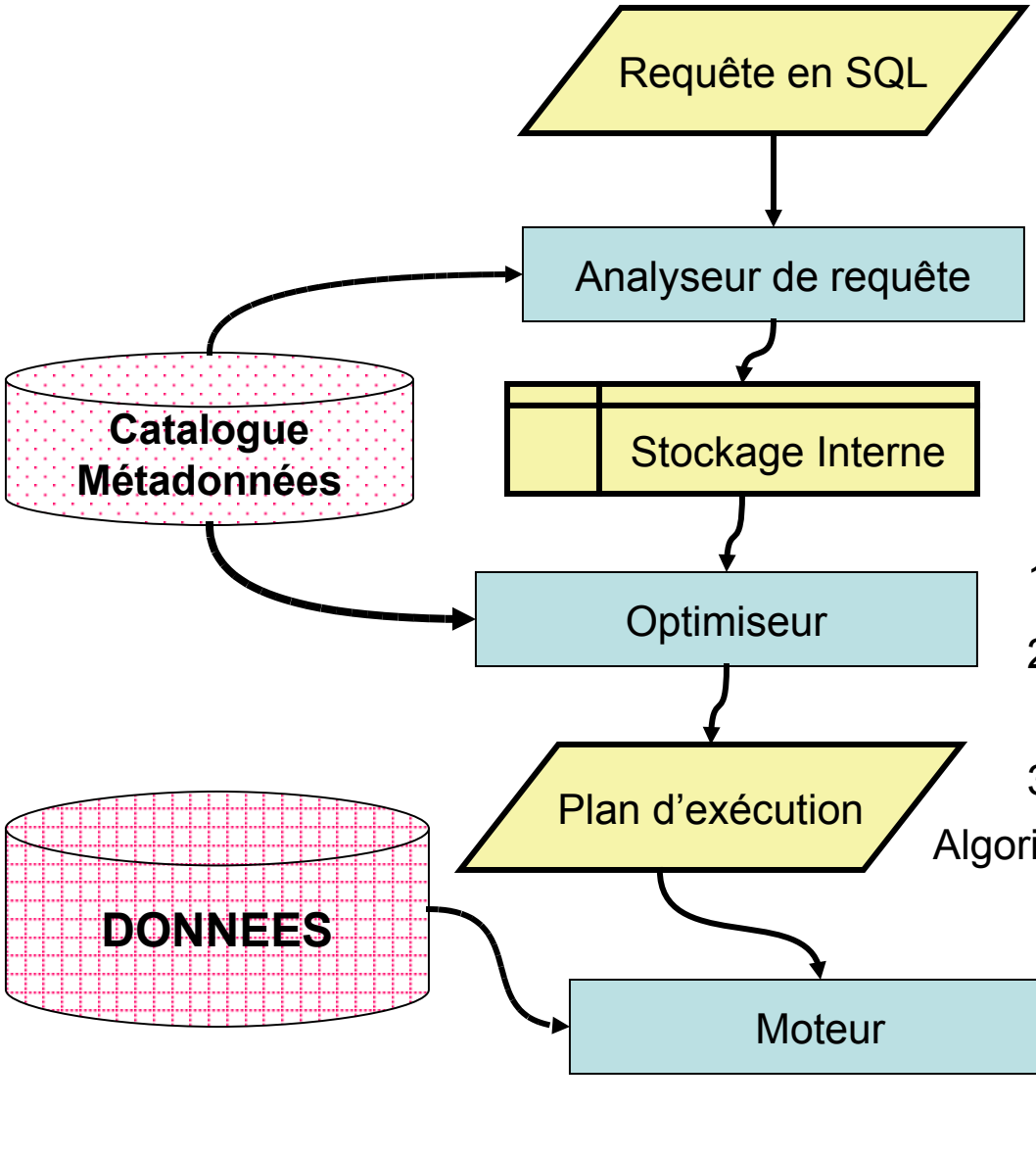
=>

10 articles créés et 70 tests de jointure.



Nécessité d'un optimiseur pour :

- Trouver une méthode de calcul performante pour toutes les méthodes de calculs possibles.
- Besoin de *statistiques* sur la base : dimension des tables et autres informations sur les données stockées
  - (cf. l'exemple les méthodes ii et iii de calcul de la jointure)



- Analyse et correction
- Vérification des droits
- Traduction en un format interne pratique

Terme de l'algèbre relationnelle, souvent traité comme un arbre, pratique à manipuler.

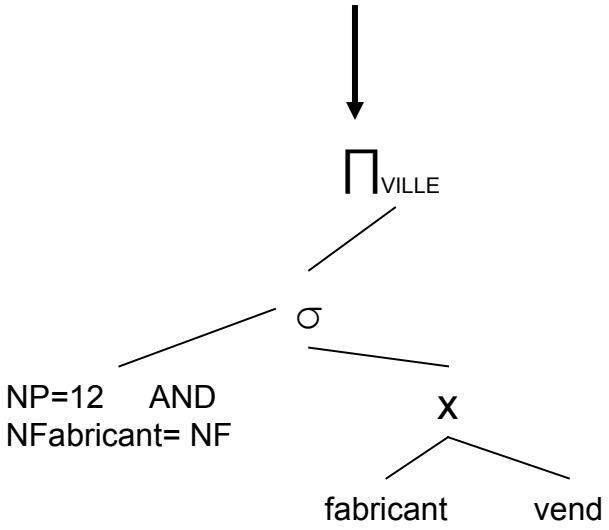
1. Transformation d'expressions.
2. Estimation des coûts de différents algos.
3. Choix de la meilleure méthode

Algorithme final de calcul du résultat.

1. Introduction 2. Fonctionnement d'un optimiseur, Exemple

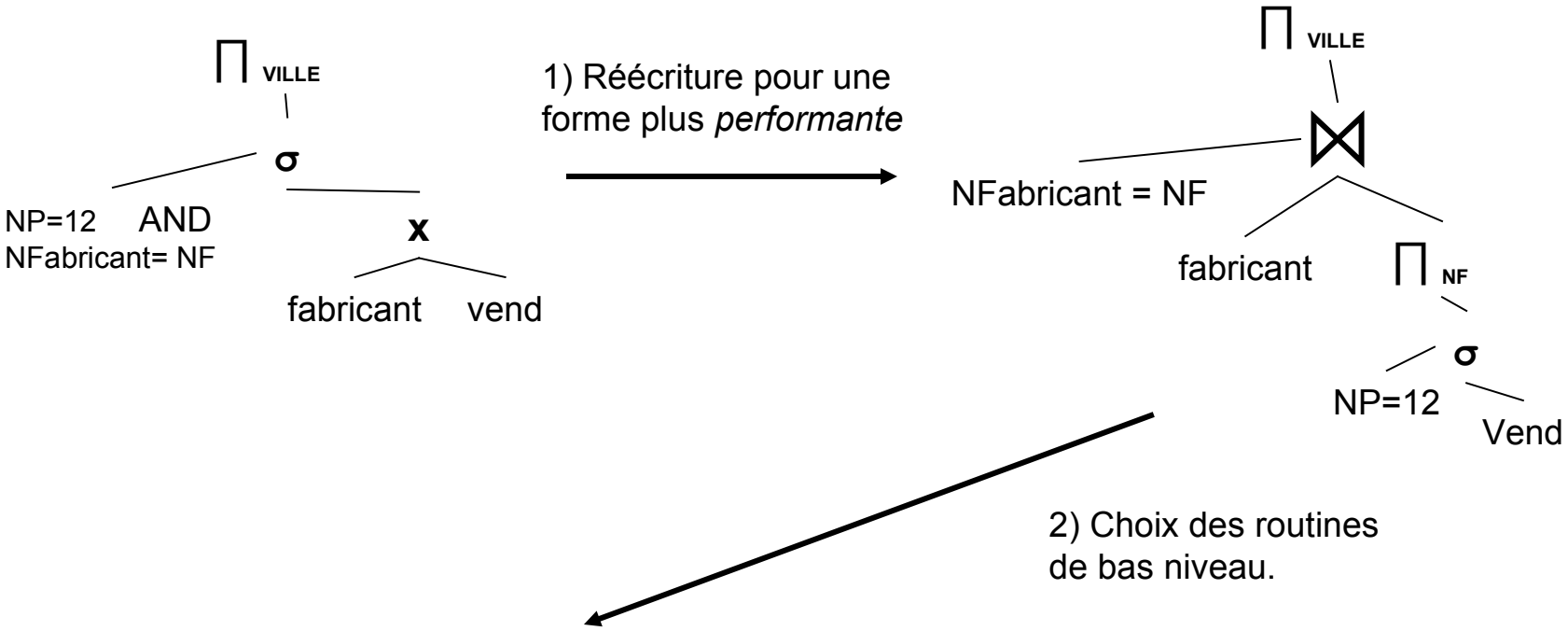
Analyse :

SELECT DISTINCT ville FROM fabricant, vend WHERE NFabricant= NF AND NP = 12 ;



1. Introduction 2. Fonctionnement d'un optimiseur, Exemple

Optimisation :



Plan d'exécution :

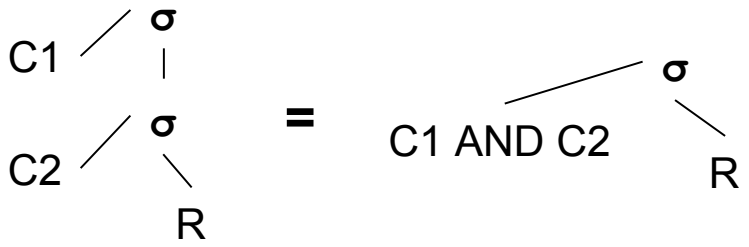
1. R1 = RECHERCHEINDEX(vend, NP,12)
2. R2 = INDEXLOOKUP(R1, fabricant, NFabricant=NF)
3. R3 = ProjectionTRI(Vend, Ville)

1. Introduction 2. Fonctionnement d'un optimiseur

- Quelques règles d'optimisation :
  - Essayer d'éliminer les enregistrement le plus tôt possible des données.
  - Minimiser la taille des relations intermédiaires.
- Choix du plan d'exécution : Pour choisir parmi tous les plans d'exécution, on a besoin d'évaluer le coût de chacun.
  - **Coût d'un plan d'exécution :** Nb d'E/S, taille des résultats intermédiaires, temps de calcul. Pas toujours facile à évaluer.
  - => Besoin de « statistiques » sur la base : informations sur la manière dont elle est remplie.
  - Les coûts sont calculés avec les informations disponibles **au moment** de l'optimisation…
  - Impossible de générer tous les plans possibles.

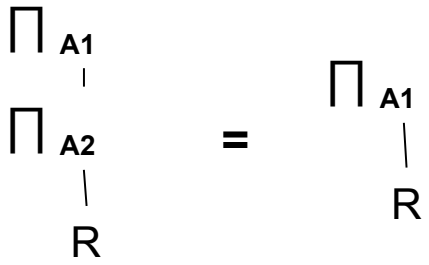
2. Réécriture de requêtes

- **SELECT(SELECT(R,C1),C2) = SELECT(R, C1 and C2)**



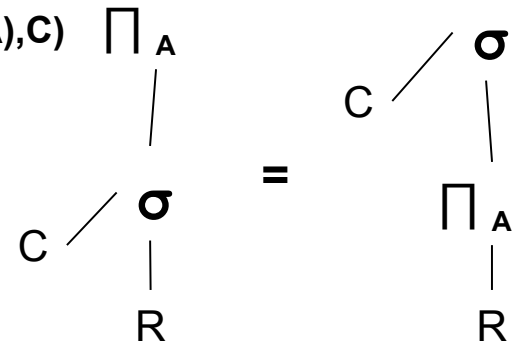
- **PROJECTION(PROJECTION(R,A2),A1) = PROJECTION(R,A1)**

Avec **A1** ⊆ **A2**



- **PROJECTION (SELECT(R,C),A) = SELECT(PROJECTION(R,A),C)**

La condition C ne porte que sur des attributs de A



2. Réécriture de requêtes

- Distributivité de la projection

- Commutativité/Associativité de l'union, intersection, produit cartésien, jointure.

- Expression scalaires :
    - Optimisation classique des compilateurs.

2. Réécriture de requêtes

- Expressions booléennes.
  - Transformation en forme normale conjonctive.

    Exemple : A > 2 OR (C=D AND E < F) devient …
    1. Calcul de la sélectivité de chaque **Ci**
    2. On évalue d'abord les plus sélectifs.

  - Parfois « duplication » de conditions. Exemple : (A et B attributs respectivement de TA et TB).

    SELECT * FROM TA, TB
    WHERE A > B and B > 3 ;

  - Utilisation des contraintes d'intégrité.
  - Utilisation des statistiques (mémorisation des extrema des attributs ?)

3. Les informations utiles à l'optimisation.

Pour chaque table :

- Cardinalité.
- Informations sur le stockage (nb de paquets ?).

Pour chaque attribut :

- Nb de valeurs distinctes.
- Extrema
- Index ?

Pour chaque index :

- Taille de l'index, nombre de niveaux pour les hierarchiques.
- Valeurs les plus fréquentes et leur fréquence.

Etc….

3. Les informations utiles à l'optimisation.

Le contenu du catalogue et la richesse des métadonnées dépendent du SGBD et peuvent être paramétrables.

- Plus de statistiques gérées = mises à jour couteuses.
- Plus de statistiques gérées = optimisation plus performante.

Ces statistiques peuvent n'être pas calculées en permanence.

- Statistiques périmées = mauvaise optimisation.
- => recalculer les statistiques régulièrement.

Attention aux requêtes compilées.

- Le contenu de la base (et les statistiques) a pu beaucoup changer depuis le calcul du plan d'exécution.
- => recompiler la requête si modification(s) sérieuse(s) de la base.

4. Différents algorithmes de bas niveau de calcul de jointures.

- On fait la jointure R ⋈ S, le champ C est en commun.
  Jointure naturelle : la condition de jointure est R.C=S.C.
- Une grosse partie de ce qui suit est transposable si la jointure n'est pas naturelle.
- R(i) : i^{eme} tuple de la table R .
- |R| : cardinalité de R.
- On s'intéresse à :
  - Nb d'E/S.
  - Nb de Test.

4. Calcul de jointures. a. Force brute

Algorithme de force brute :

```
Pour i de 1 à |R| faire
        Pour j de 1 à |S| faire
                Si R(i).C =S(j).C alors
                        ajouter R(i)&S(j) au Résultat.
                Finsi
        Finpour
FinPour
```

4. Calcul de jointures. a. Force brute

Accès et nb de tests :

- **|R|** lectures de tuples de R
- **|R|.|S|** lectures de tuples de S.
- **|R|.|S|** tests d'égalité.

Taille du résultat : *(variable en règle générale)*

Souvent l'un est une clé étrangère

=> **|Resultat| = |Re|** si Re est la relation où C est clé étrangère.

4. Calcul de jointures. a. Force brute

Nombre d'entrées sorties : R tient sur Pr pages, et S sur Ps pages

```
Boucle sur R
    Boucle sur S
    finBoucleSurS
finBoucleSurR
```

```
Boucle sur S
    Boucle sur R
    finBoucleSurR
finBoucleSurS
```

Nb pages lues : **Pr + |R|.Ps** **Ps + |S|.Pr**

Exemple : |R|= 100, Pr =100, |S| = 10000, Ps =1000

=>

4. Calcul de jointures. b. Recherche d'index

Algorithme de recherche d'index : On suppose qu'il existe un index sur **S.C**

Pour **i** de **1** à **|R|** faire
    Utiliser l'index pour trouver les articles de **S** t.q. **S.C =R(i).C.**
        (soient **x1**, **x2**, … **xk**, les **k** valeurs trouvées).
    Pour **j** de **1** à **k** faire
        ajouter **R(i)&S(xj)** au Résultat.
    Finpour
FinPour

4. Calcul de jointures. b. Recherche d'index

Accès et nb. de tests :

- **|R|** lectures de tuples de **R**.
- **|R|** recherches dans l'index (dépend de l'organisation de l'index et du nombre d'articles concernés) on peut considérer **0(log(|S|)+k)** pour un arbre B, pour **k** tuples concernés.
- => en **O(|R|.log(|S|)+K).** Où **K** est la taille du résultat (nombre d'articles concernés pour toute la jointure).

4. Calcul de jointures. b. Recherche d'index

Nombre de chargement de pages : **R** tient sur **Pr** pages.

- **Pr** accès disques pour parcourir la table **R**.
- **|R|** recherches dans l'index (chacune dépend de la taille, de l'organisation de l'index et du nombre d'articles concernés).
  - Pour un arbre B, en **$O(\log_m(|S|)+k$** entrées sorties pour **k** tuples concernés (répartis au pire dans **k** paquets différents)
  - Si l'index tient sur **Pi** (faible) pages mémoires, il peut rester en mémoire tout le long du calcul de la jointure.
- => de **O(Pr + Pi + K)** à **O(Pr + |R|.log(|S|)+K)** entrées/sorties suivant le cas**.** (**K** est la taille du résultat (nombre d'articles concernés pour toute la jointure).

4. Calcul de jointures. c. Recherche par hachage

Algorithme de jointure par hachage : On suppose que la table S est hashée sur l'attribut **C** en utilisant la fonction **hash**

Pour **i** de 1 à **|R|** faire
    h = hash(R(i).C) ;
    Récupérer les tuples de S présents à l'entrée **h.**
        (Soient **x1, x2, …xk** les **k** tuples concernés).
    Pour **j** de **1** à **k** faire
        Si **S(xj).C = R(i).C** alors
            ajouter **S(xj)&R(i)** au résultat.
        finSi
    finPour
finPour

4. Calcul de jointures. d. Trifusion

Les tables R et S sont toutes deux triées suivant l'attribut C.
Dans ces deux tables C est sans doublon.

Algorithme de trifusion classique.

Nombre de paquets lus : **Pr + Ps**
Nombre de tests : en **O(|R|+|S|)**

4. Calcul de jointures. e. Hachage

On va hacher dans une même table les tables R et S, si collision entre deux éléments, on vérifie la condition.

Algorithme :

Construire une table de hachage avec R.

Appliquer l'algorithme de recherche par hachage.

5. Divers 1. Traitement de l'opérateur IN.

$$\sigma_{A \in S}(R)$$

La liste S peut être une sous requête dépendant d'un attribut de R.

S dépendant ou non de R
Boucle externe sur R :

Pour **i** de 1 à **|R|** faire
    calculer **S**
    chercher dans **S**, le premier **R(i).A**
    si existant, alors
        rajouter **R(i)** au résultat.
    finSi
finPour

S indépendant de R
Boucle externe sur S :

Calculer S.
Pour **i** de **1** à **|S|** faire
    - chercher dans **R** sur **C**, la valeur **S(i)** et rajouter au résultat chacun des tuples de **R** trouvés.
finPour

5. Divers 1. Traitement de l'opérateur IN.

$$\sigma_{A \in S}(R)$$

Si S est indépendant de R, on peut sortir son calcul de la boucle.

Nombre d'entrées sorties ?

Boucle externe sur R
(calcul de S non compris) :

Pr + |R|.recherche_dans_S.

- Sans index sur S.
  Pr + |R|.Ps
- Si il y a un index sur S (si créé lors du calcul de S, rajouter le coût de son calcul)
  en **O(Pr + |R|.log(|S|))**

Boucle externe sur S
(calcul de S non compris) :

Ps + |S|.recherche_dans_R

- Sans index sur R.
  Ps + |S|.Pr
- Si il y a un index sur R
  en **O(Ps + |S|.log(|R|)+K)** où **K** est la taille de la table résultat.

5. Divers 2. Recherche de sous expressions communes.

a) Au cours de plusieurs sessions ou requêtes différentes.

Exemple :

Facture (NFacture, NClient, REmise)

Produit(NProduit,Prix)

LigneFacture(NFacture, NProduit, Quantité)

Si l'on a souvent besoin de

(NFacture, NClient, PrixTotal) ?

=> Le stocker en permanence, mécanisme de vue concrète (cf plus tard).

5. Divers 2. Recherche de sous expressions communes.

b) Dans une seule requête.

Exemple :

Ville(CP, Population)

Fabricant(NFabricant, CP, CA)

Magasin(NMagasin, CP, CA)

Chiffre d'affaire des entreprises situées dans les villes de plus de 100000 habitants.

Détection automatique par l'optimiseur.

(Mécanisme classique d'optimisation).

5. Divers 3. Autres

Optimiser le plan d'exécution :

- Mécanismes propres aux bases de données.
  - Problématique particulière de la recherche de valeur et utilisation des index
  - Estimation des tailles de résultats, etc…
- Mécanismes communs à la compilation.
  - Recherche de sous expressions communes.
  - Sortie d'invariant de boucle,
  - Etc.