Systèmes de Gestion des Bases de Données

Traitement des requêtes dans les SGBD-R

Nacer Boudjlida

Traitement des requêtes dans les SGBD relationnels

- Evaluation des requêtes :
  - Données de la base vers la mémoire centrale
  - Traitement
  - "Retour" dans la base si mise à jour
- Peut nécessiter la création de tables intermédiaires
- Volume des transferts et des tables de travail fonction des tailles des relations mises en jeu
- *Objectif d'un optimiseur* : Réduction des volumes et du temps

Traitement des requêtes : Besoins d'optimiser ?

- Temps d'exécution d'une requête "anormal" % taille des relations, existence d'index
- Une requête s'exécute plus lentement que des requêtes similaires
- $\nearrow$ temps d'exécution d'une requête
- Temps d'exécution d'une requête en tant que procédure $>$ celui de son exécution en tant que requête
- Plan d'exécution utilisant un parcours de relation au lieu d'utiliser un index

Traitement des requêtes : Origines des mauvaises performances ?

- "Vieilles" statistiques sur la distribution des données
- Inexistence d'index appropriés
- Index en cours d'utilisation pour accéder à une table volumineuse
- Clause *where* conduisant au choix d'une mauvaise stratégie
- Procédure non re-compilée après changements significatifs
- etc.

Traitement des requêtes : Les méthodes

1. Syntaxique :
   - Fondement : Heuristiques + Propriétés de l'algèbre
   - Transformation d'arbres (algèbre) ou de graphes (calcul relationnel)
2. Estimation de coûts d'exécution de diverses stratégies d'évaluation
3. Sémantique : Exploitation des contraintes d'intégrité

- Méthodes non exclusives : 1 et 2 souvent combinées

Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'Optimisation

I. Transformation d'arbres algériques

- Heuristiques : $\searrow$ taille des opérandes des opéraions les plus coûteuses
  1. "Descente" des sélections : $\searrow$ les relations "en hauteur"
  2. "Descente" des projections : $\searrow$ les relations "en largeur"
- [Phrase SQL $\longrightarrow$] Arbre algébrique :
  - *Feuilles* : Relations
  - *Racine* : Résultat de la requête
  - *Nœuds internes* : Opérations de l'algèbre
  - *Arcs "entrants"* : Opérandes
  - *Arcs "Sortants"* : Résultat de l'évaluation d'un nœud
- Evaluation de "bas en haut" (gauche-droite ou droite-gauche)

I. Transformation d'arbres algériques : Exemple

- *Personne*(id#, nom, prenom, date_naisssance, adresse, equ#)
- *Equipe*(equ#, nom_equ, resp_equ#)
- *Projet*(proj#, nom_proj, lieu_proj, equ#)
- *Travaille_sur*(id#, proj#, taux)
  - *Personne.equ#* : Equipe de rattachement
  - *Projet.equ#* : Equipe en charge du projet
  - *Travaille_sur.taux* : Part de temps sur un projet
- *Requête* :
  Personnes nées après 1962 et travaillant sur le projet "BDD"

I. Transformation d'arbres algériques : Exemple

- En SQL :

```
SELECT  nom
FROM    Projet, Travaille_sur, Personne
WHERE   Projet.nom_proj = 'BDD'
AND     Personne.id# = Travaille_sur.id#
AND     Personne.date_naissance > 'dec-31-1962'
AND     Travaille_sur.proj# = Projet.proj#
```

- *Représentation dite canonique* :
  - *Relations de la clause FROM* $\longrightarrow$ Produit cartésien (X)
  - *WHERE* $\longrightarrow$ Sélection (σ), nœud père du sous-arbre X
  - *SELECT* $\longrightarrow$ Projection (Π) en racine de l'arbre

Transformation d'arbres algériques : Arbre initial (0)

Transformation d'arbres : Arbre 1

- *"Descente" des sélections*

Transformation d'arbres : Arbre 2

- *Permutation de Personne et de Projet* (cf. notion de sélectivité) : **Si mélange méthodes syntaxique et estimation de coûts**.

Transformation d'arbres : Arbre 3

- *Introduction de jointures*

Transformation d'arbres : Arbre 4

- *Introduction de projections* : ne "faire remonter" que les attributs utiles

Transformation d'arbres : Sur l'arbre initial (Arbre 0)

- *Projet* : 20 tuples de 100 caractères
- *Travaille_sur* : 100 tuples de 50 caractères
- *Personne* : 500 tuples de 100 caractères
- *Deux produits cartésiens* : 10^6 tuples de 250 caractères !

Transformation d'arbres : Sur l'arbre final (Arbre 4)

- *1ère jointure* :
  - Une relation à une colonne et probalement à un n-uplet
  - Relation *Travaille_sur* : deux colonnes
- *2ème jointure* :
  - Une relation à une colonne (sous-arbre gauche)
  - Une relation à 2 colonnes (sous-arbre droit) réduite aux seuls n-uplets satisfaisant la sélection

Transformation d'arbres : Règles de transformation

- ***(R1)*** Décomposition des expressions de sélection.
  $Select(R,\ C_1 \wedge C_2 \wedge \cdots \wedge C_{n-1} \wedge C_n) \longrightarrow$
  $Select(Select(\cdots Select\ (Select(R,\ C_n),\ C_{n-1})),\ \cdots),\ C_2),\ C_1)$
- ***(R2)*** Commutativité de la sélection.
  $Select(Select(R,\ C_2), C_1) \longrightarrow Select(Select(R,\ C_1),\ C_2)$.
- ***(R3)*** Restriction de la liste de projections.
  $Proj_{L1}(Proj_{L2}(\cdots\ (Proj_{Ln}(R))\ \cdots) \longrightarrow Proj_{L1}(R)$.
- ***(R4)*** Commutation de la sélection et de la projection.
  Si C ne porte que sur les A_i :
  $Proj_L(Select(R,\ C)) \longrightarrow Select(Proj_L(R),\ C)$

Transformation d'arbres : Règles de transformation

- ***(R5)*** Commutativité de la jointure (et du produit cartésien).
  $Join_C\ (R,\ S) \longrightarrow Join_C(S,\ R)$.
- ***(R6)*** Commutation sélection-jointure (ou produit cartésien).
  - ***(R61)*** Si les attributs de la condition C de sélection n'appartiennent qu'à une des relations de la jointure, par exemple R, alors :
    $Select(Join_E(R,\ S),\ C) \longrightarrow Join_E(Select(R,\ C),\ S)$.
  - ***(R62)*** Sinon, si la condition C peut se ré-écrire en $C_1 \wedge C_2$ où :
    - C_1 ne porte que sur des attributs de R
    - $C_1\ C_2$ ne porte que sur ceux de S, alors :

    $Select(Join_E(R,\ S),\ C) \longrightarrow$
    $Join_E(Select(R,\ C_1),\ Select(S,\ C_2))$

Transformation d'arbres : Règles de transformation

- ***(R7)*** Commutation projection-jointure (produit cartésien).
  - ***(R71)*** $Proj_L(Join_C(R,\ S)) \longrightarrow$
    $Join_C(Proj_{L_1}(R),\ Proj_{L_2}(S))$
    1. Si attributs de C = attributs de L
    2. Si $L = L1||L2$ avec $L_1 = Liste(A_i),\ A_i$: attributs de R et $L_2 = Liste(B_j),\ B_j$: attributs de S
  - ***(R72)*** Si la condition de jointure comporte des sous-listes L_3 d'attributs de R et L_4 d'attributs de S n'appartenant pas à L :
    1. Les ajouter aux projections "internes"
    2. Appliquer une projection "externe" sur L

    $Proj_L(Join_C(R,\ S)) \longrightarrow$
    $Proj_L(Join_C(Proj_{L_1,\ L_3}(R), Proj_{L_2,\ L_4}(S)))$.

Transformation d'arbres : Règles de transformation

- ***(R8)*** Commutativité des opérations ensemblistes ($\cup$, $\cap$).
- ***(R9)*** Associativité : jointure, produit cartésien, union et intersection.
  $((R\ op_i\ S)\ op_j\ T) \longrightarrow (R\ op_j (S\ op_i\ S))$ où R, S, T sont des relations et op_i, op_j des opérateurs parmi ceux cités
- ***(R10)*** Commutation de la sélection et des opérations ensemblistes.
  (Select((R op S), C)) $\longrightarrow$ (Select(R, C) op Select (S, C)) où op est l'opérateur d'union, d'intersection ou de différence

Transformation d'arbres : Règles de transformation

- ***(R11)*** Commutation de la projection et des opérations ensemblistes.
  $(Proj_L(R\ op\ S)) \longrightarrow ((Proj_L\ (R))\ op\ (Proj_L(S)))$.

- Autres :
  - Equivalences logiques
  - Autres propriétés des opérateurs algébriques

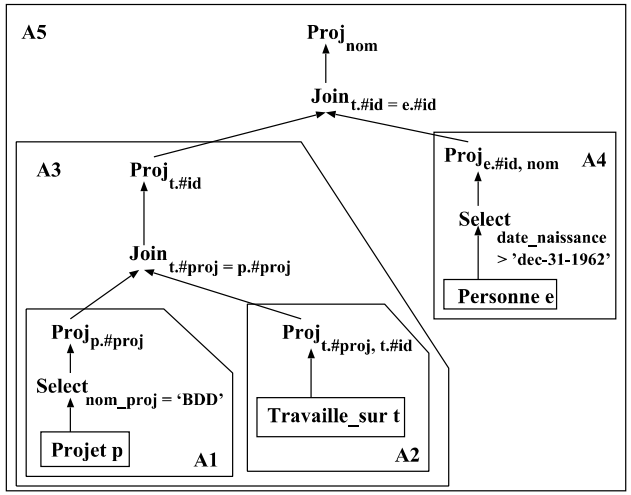
Transformation d'arbres : Ebauche d'algorithme

1. Transformer toute expression de sélection conjonctive en une "cascade" de sélections (Règle R1)
2. "Descendre" les sélections le plus bas possible ((R2), (R4), (R6), (R10)).
3. Ordonner les feuilles de l'arbre de sorte que les sélections les plus restrictives soient évaluées en premier ((R8), (R9))
   - **Etape à n'exécuter que si** combinaison méthode syntaxique et méthode par estimation de coûts ;
   - "Plus restrictives" = produisant les plus petites relations ;
   - cf. distribution des valeurs, index (dictionnaire) ;
   - cf. notion de sélectivité, plus loin.

Transformation d'arbres : Ebauche d'algorithme

4. (Produit cartésien; Sélection) $\longrightarrow$ Jointure, où la condition de jointure est la condition de sélection.
5. "Fragmenter" et "faire descendre" le plus bas possible les listes de projection, en créant de nouvelles projections, si besoin est (Règles (R3), (R4), (R7), (R11)) ,
6. Identifier et ordonner les sous-arbres qui représentent des groupes d'opérations pouvant être exécutés par une seule routine du SGBD.

Transformation d'arbres : Exemples de plan (Etape 6)



Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'optimisation

II. Optimisation de graphes de requêtes

- Technique dite de décomposition de requêtes
- Introduite dans Ingres/QUEL (Variables n-uplets)
- *Graphe de requêtes* :
  - *Sur les arcs* : Conditions de jointure des nœuds reliés
  - *Nœuds* : Variables de tuples ou constantes de la requête
  - *Arcs "'conditions de sélection"'* : relient des nœuds de constantes aux nœuds des variables impliquées dans les conditions.
- Requête à n variables $\longrightarrow$ m requêtes, plus simples, à (n-1) variables.

Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'Optimisation

III. Optimisation par estimation de coûts

- Minimiser une fonction côut, avec comme facteurs :
  - Accès disque,
  - Coût du traitement en mémoire centrale,
  - Taille des n-uplets, des relations,
  - Index : Existence, nombre de niveaux,
  - Coût de communication (requête, résultats) entre sites (architectures client/serveur, par exemple),
  - etc.
- Utilisation effective (souvent) des coûts des accès

III. Optimisation par estimation de coûts

- *Informations du dictionnaire*
  - Nombre exact ou estimé de tuples et de blocs physiques par relation ;
  - Nombre de niveaux d'index ;
  - Nombre de valeurs distinctes par attribut, dans les cas où un index ou un cluster existe : permet d'estimer le nombre moyen d'enregistrements qui satisfont une sélection (*sélectivité* ou *cardinal de sélection* d'un attribut A).

    * A est clé : *s = 1*
    * A n'est pas clé : *s = Card(R)/Nombre de valeurs de A*.

III. Optimisation par estimation de coûts

- *Sélectivité de la jointure* : $sj = Card((R \bowtie_{Cond} S))/Card(R \times S)$
  - *sj* = 1 si pas de condition de jointure
  - *sj* = 0 si aucun tuple ne vérifie *Cond*.

- *Cas de l'équi-jointure* : (*Condition du type R.A = S.B*) Cas particuliers :
  1. A est clé de R : $Card(R \bowtie S) \leq Card(S)$
     - $sj \leq Card(S)/Card(R) \times Card(S)$
     - $sj \leq 1/Card(R)$
  2. De même, si B est clé de S : $sj \leq 1/Card(S)$.
- D'où la taille estimée de $R \bowtie S$: *sj* × *Card(R)* × *Card(S)*.

Exemples d'estimation de l'équi-jointure

- br : nombre de blocs de R,
- bs : nombre de blocs de S
- k : facteur de blocage de la relation résultat

1. *Jointure par boucles imbriquées*
   - R dans la boucle extérieure ; *sj* donné ; 2 buffers

   - Coût estimé : *br + (br * bs) + ((sj * Cardinal(R) * Cardinal(S))/k)*
   - Dernier facteur : Coût de l'écriture du résultat.

Exemples d'estimation de l'équi-jointure

2. *Jointure par tri-fusion*
   - Si relations déjà triées : *Coût = br + bs*
   - Sinon, *Coût* = $br + bs + l * (br * \log_2 br + bs * \log_2 bs))$.
     - $(l * b * \log_2 b)$: Approximation du tri
     - b : nombre de blocs
     - l : facteur constant.

IV. Optimisation sémantique

- *Contrainte d'intégrité* Formule logique CI
- *Expression de sélection* E
- *Pas d'accès à la base si* $\neg(E \wedge CI)$
- *Exemple* :
  - CI : Tous les vols long courrier partent de Paris-Roissy
  - E : Vols long courrier partant de Paris-Orly ?
- Méthode non implantée (démonstration automatique)

Optimisation de requêtes : SGBD Sybase

- Génération de plans avec/sans exécution de requête
- Exemple : Visualisation d'un plan sans exécution

  SET SHOWPLAN ON
  SET NOEXEC ON
  Requête dont on veut examiner le plan.
- Plans d'exécution et procédures stockées :
  - Exécution avec *WITH RECOMPILE* : MàJ du plan stocké
  - Création avec *WITH RECOMPILE* : Génération systématique
- *Mise à jour* des informations sur la distribution des valeurs des clés des index :

UPDATE STATISTICS NomDeRelation [NomIndex]

Optimisation de requêtes : SGBD Oracle

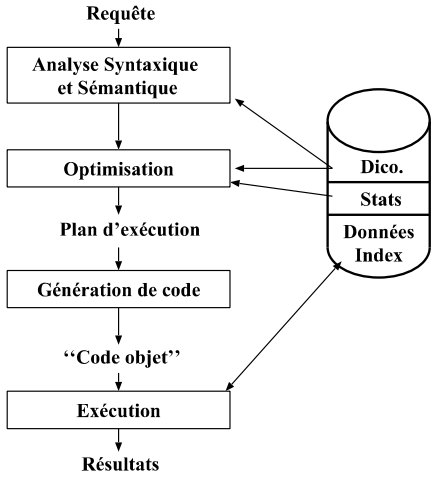
- Commande *EXPLAIN PLAN* : obtenir le plan d'exécution
- Plan rangé dans *PLAN_TABLE* ou dans une relation créée préalablement à l'exécution de *EXPLAIN PLAN* (clause *INTO* de *EXPLAIN PLAN*)

  EXPLAIN PLAN
  [*SET STATEMENT_ID* = identification_plan]
  [*INTO* [nom_utilisateur.]nom_de_relation] *FOR* requête_SQL
- ANALYZE {index| table | cluster} {COMPUTE | ESTIMATE} STATISTICS

Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'Optimisation

V. Optimisation : Processus général



Traitement des requêtes : Processus général

1. Normalisation des prédicats
2. Analyse sémantique (Requête = Graphe connexe)
3. Simplification des formules logiques
4. Mise sous forme d'arbre relationnel

1/4) Normalisation des prédicats

- *Forme normale conjonctive* : conjonction de disjonctions
  $(p_1 \vee p_2 \vee \cdots \vee p_n) \wedge \cdots \wedge (q_1 \vee \cdots \vee q_m)$
- *Forme normale disjonctive* : disjonction de conjonctions
  $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \vee \cdots \vee (q_1 \wedge \cdots \wedge q_m)$
- *Formules sans quantificateurs* :
  - Commutativité, Associativité de la conjonction et de la disjonction
  - Distributivité $(\vee, \wedge)$, $(\wedge, \vee)$
  - $\neg(p_1 \wedge p_2) \longleftrightarrow \neg p_1 \vee \neg p_2$
  - $\neg(p_1 \vee p_2) \longleftrightarrow \neg p_1 \wedge \neg p_2$
  - $\neg(\neg p) \longleftrightarrow p$
- *Formules avec quantificateurs* : mise sous forme *prenex*

- *Forme disjonctive* :
  - Requête traitée comme une union de sous-requêtes
  - Risque de calcul redondant
- *Forme conjonctive* : généralement plus de "ET" que de "OU"
- *Exemple* :

  select libelle from produit p, stock s
  where p.prod# = s.prod# and s.adr = "Nancy"
  and (s.qte = 1000 or s.qte=200)

  - *Forme disjonctive* :

    (p.prod# = s.prod# $\wedge$ s.adr = "Nancy" $\wedge$ s.qte = 1000)
    $\vee$ (p.prod# = s.prod# $\wedge$ s.adr = "Nancy" $\wedge$ s.qte = 2000)
  - *Forme conjonctive* :

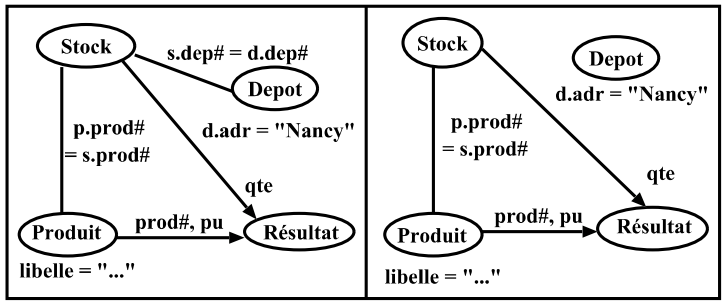
    p.prod# = s.prod# $\wedge$ s.adr = "Nancy" $\wedge$ (s.qte = 1000 $\vee$ s.qte=2000)

2/4) Analyse

- *Dictionnaire* : Relations, attributs connus
- *Typage des expressions*
- *Correction sémantique* : pas de sous-requête isolée
  - *Calcul relationnel* : impossible à determiner
  - *Requêtes sans $\vee$ ni $\neg$* : graphe connexe
- *Graphe de requête* :
  - *Nœud* : résultat ou opérande
  - *Arcs entre nœuds non résultats* : Jointure
  - *Arcs d'extrêmité nœud résultat* : Projection
  - *Nœud non résultat* peut être labellé par une expression de sélection ou une auto-jointure

- *Exemple* :

```
select p.prod#, p.pu, s.qte
from produit p, stock s, depot d
where p. prod# = s.prod# and s.dep# = d.dep#
   and s.qte >= 0 and d.adr = "Nancy" and libelle = "..."
```



3/4) Simplification d'expressions logiques

- *Cas de l'utilisation de vues*

```
create view V
as      select * from produit
        where pu > 100.0 and pu < 200.0
```

- *Requête* :

```
select * from V
where pu < 200.0
```

- *Après ré-écriture* :

```
select * from produit
where pu > 100.0
   and pu < 200.0 and pu < 200.0
```


3/4) Simplification d'expressions logiques (suite)

- *Elimination de la redondance* : règles d'idempotence
  - $p \wedge p \longleftrightarrow p; p \vee p \longleftrightarrow p$
  - $p \wedge Vrai \longleftrightarrow p; p \wedge Faux \longleftrightarrow Faux$
  - $p \vee Vrai \longleftrightarrow Vrai; p \vee Faux \longleftrightarrow p$
  - $p \wedge \neg p \longleftrightarrow Faux; p \vee \neg p \longleftrightarrow Vrai$
  - $p_1 \wedge (p_1 \vee p_2) \longleftrightarrow p_1$
  - $p_1 \vee (p_1 \wedge p_2) \longleftrightarrow p_1$

3/4) Simplification d'expressions logiques (suite et fin)

- *Exemple* :

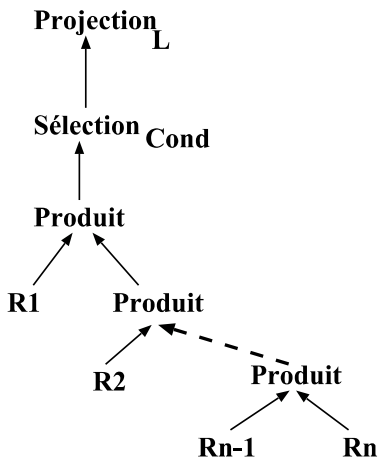
  select libelle from produit
  where NOT libelle = "K7Cr"
  and (libelle = "K7Cr" or pu = 20.0)
  and NOT pu = 20.0
  and prod# = 4

  se simplifie en :

  select libelle from produit
  where prod# = 4

4/4) Mise sous forme d'arbre algébrique

SELECT L
FROM R1, R2, ..., Rn
WHERE Cond



Traitement des requêtes sous Oracle

- Etapes "classiques" dans le processus de traitement des requêtes :
  1. Analyse,
  2. Optimisation,
  3. Génération de plans d'exécution,
  4. Exécution.

Processus de Traitement des requêtes sous Oracle

- Deux stratégies d'optimisation :
  1. A base de règles (*Rule Based Optimization*, *RBO*) : $\simeq$ optimisation d'arbres algébriques
  2. A base de coûts : *Cost Based Optimization* (*CBO*).
- Choix de la stratégie : automatique ou imposé pour une requête, une session ou une instance Oracle,
- Influencer la génération de plans : Directives d'optimisation dans les requêtes,
- Ré-utiliser des plans : cf. *outlines* (non traités ici).