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
- ↗ 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

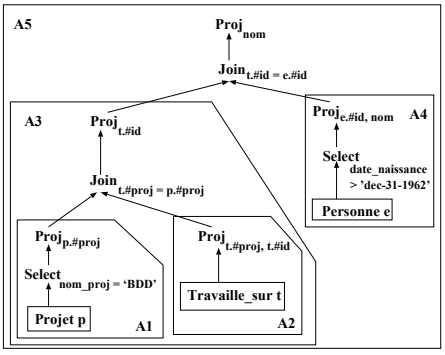
Transformation d'arbres : Ebauche d'algorithme

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

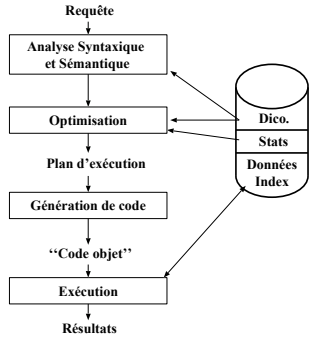
- Commande *EXPLAIN PLAN* : obtenir le plan d'exécution
- Plan rangé dans *PLAN_TABLE* ou dans une relation créée préalablement à l'exécution de *EXPLAIN PLAN* (clause *INTO* de *EXPLAIN PLAN*)

  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# ∧ s.adr = "Nancy" ∧ s.qte = 1000)
    ∨ (p.prod# = s.prod# ∧ s.adr = "Nancy" ∧ s.qte = 2000)
  - *Forme conjonctive* :

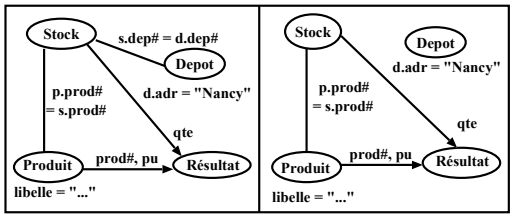
    p.prod# = s.prod# ∧ s.adr = "Nancy" ∧ (s.qte = 1000 ∨ s.qte=2000)

2/4) Analyse

- *Dictionnaire* : Relations, attributs connus
- *Typage des expressions*
- *Correction sémantique* : pas de sous-requête isolée
  - *Calcul relationnel* : impossible à determiner
  - *Requêtes sans ∨ ni ¬* : 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* :

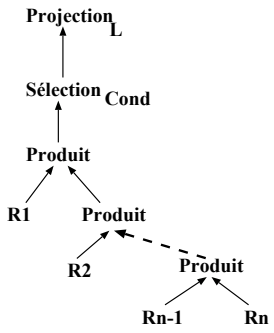
```
select libelle from produit
where NOT libelle = "K7Cr"
      and (libelle = "K7Cr" or pu = 20.0)
      and NOT pu = 20.0
      and prod# = 4
```

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.

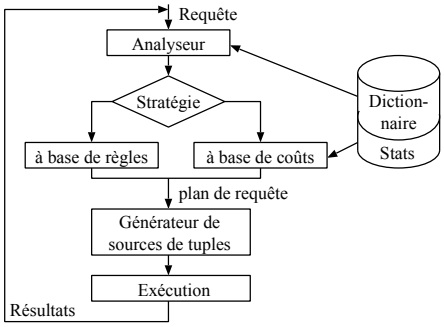
Processus de Traitement des requêtes sous Oracle

- 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).

Traitement des requêtes sous Oracle : Plan

1. Architecture du compilateur de requêtes et structure de stockage des plans d'exécution (p. 50)
2. Processus et principales étapes de traitement des requêtes (p. 59)
3. Optimisation à base de coûts (p. 86)
4. Optimisation à base de règles (p. 105)
5. Directives d'optimisation (p. 108)

1/5. Architecture de l'optimiseur et plans d'exécution



1/5. Architecture de l'optimiseur et plans d'exécution

1. *Analyseur* : décomposition de la requêtes en un ensemble de composants emboîtés ou reliés (*blocs de la requête*)
   - Vue, requête imbriquée $\longrightarrow$ blocs distincts de celui de la requête (ou pas)
2. *Générateur de sources de tuples* : Engendrer un plan d'exécution étant donné le plan optimal ($\simeq$ arbre algébrique) rendu par l'optimiseur :
   - Noeuds opérations "internes" à Oracle ($\neq$ opérateurs algébriques) :
     - Provenance = mode d'accès aux tuples et type d'algorithme de jointure choisis
     - Exemples : tri (*sort*), fusion (*merge*)
   - Source de tuples $\simeq$ feuille d'un arbre ou relation résultant de l'évaluation d'un sous-arbre.

1/5. Exemple de plan d'exécution

- Hypothèse : Aucun index
- Requête :

  select p.libelle, s.qte *from* produit p, stock s
  where p.prod# = s.prod#;

- Plan d'exécution :

1/5. Plans d'exécution

- Demande de génération : *explain plan . . . for* instruction SQL
- Plan engendré :
  - par défaut, dans *plan_table*
  - ou dans une table de même schéma (*explain plan . . . into . . .*)
- *plan_table* créée par le script *utlxplan.sql* (généralement sous *$OraHome\rdbms\admin*).
- Examen du plan :
  1. *select . . . from plan_table where . . .*
  2. utilitaire *utlxpls* : traitements en série
  3. utilitaire *utlxplp* : exécutions parallèles

1/5. Plans d'exécution : Des colonnes de *plan_table*

- *id :* numéro d'étape du plan ;
- *parent_id :* numéro de l'étape qui utilise les résultats de l'étape *id* ;
- *position :* rang d'une étape parmi les étapes "filles" d'une même étape "mère" ;
- *cardinality :* nombre estimé de tuples accédés par l'opération ;
- *operation :* nom de l'opération interne réalisée dans l'étape ;
- *object_name :* objet concerné par l'opération ;
- *cost :* coût estimé de l'opération (si optimisation basée sur les coûts) ;
- *options :* "variante" d'opération utilisée pour exécuter *operation*, etc.

1/5. Plans d'exécution : Exemple

1/5. Plans d'exécution : Exemple

```
Id P_id    Operation       Objet    Options
-- ---- ----------------- -------- ---------
 0       SELECT STATEMENT
 1    0  MERGE JOIN
 2    1  SORT                        JOIN
 3    2  TABLE ACCESS       STOCK    FULL
 4    1  SORT                        JOIN
 5    4  TABLE ACCESS       PRODUIT  FULL
6 ligne(s) sélectionnée(s).
```

- cf. figure exemple p. 52

1/5. Plans d'exécution

- *plan_table* : représentation tabulaire des plans d'exécution et des choix de l'optimiseur :
  - mode de parcours des objets
  - type d'algorithme de jointure, etc.
- Choix explicités dans *plan_table.operation* et *plan_table.options*

Opération	**Options**	**Commentaires**
table access	*full, by rowid, hash, ...*	Type d'accès à une table.
sort	*unique*	Tri en éliminant les doublons.
	join	Tri avant jointure par fusion.
	order by	Tri requis par la requête.
nested loops		Jointure par boucles imbriquées.
	outer	Idem pour une jointure externe.
merge join		Jointure par tri-fusion.
	outer	Idem pour une jointure externe.
	semi	Idem pour une semi-jointure.
view		Interrogation d'une vue.

2/5. Les étapes de l'optimisation

1.
2.
3.
4.
5.
6.
7.

2.1/5- Simplification et évaluation d'expressions

- cf. Traitement des requêtes dans les SGBDR
- *Simplification d'expressions logiques* : éliminer des sous-expressions communes par la mise d'expressions composées sous forme normale conjonctive $[(p_1 \vee \cdots \vee p_n) \wedge \cdots \wedge (q_1 \vee \cdots \vee q_m)]$.
- *Transformation d'expressions logiques* : règles de réécriture syntaxique et équivalences logiques.

1. Expression de constante évaluable statiquement : la remplacer par sa valeur (Exemple : *"X > 550/10"* $\longrightarrow$ *"X > 55"*).
2. *X like 'ABCD'* $\longrightarrow$ *X = 'ABCD'*, si X est de type chaîne de caractères à longueur variable.

2.1/5- Simplification et évaluation d'expressions

3. *"x between v1 and v2"* $\longrightarrow$ *"x >= v1 and x <= v2"*.
4. *"x in (V_1, ..., V_n)"* $\longrightarrow$ (*"x = V_1 or ... or x = V_n"*).
5. *"x > all (V_1, ..., V_n)"* $\longrightarrow$ (*"x > V_1 and ... and x > V_n"*)
6. *"x > all (select y ... where condition)"* $\longrightarrow$ *"not exists (select y ... where condition and x <= y)"*.
7. Idem pour des expressions comprenant *any* ou *some* :
   Exemple1 : *"x > any (y, z)"* $\longrightarrow$ *"x > y or x > z"*
   Exemple2 : *"x > any (select y ... where Condition)"* $\longrightarrow$ *"exists(select y ... where Condition and x > y)"*.

2.1/5- Simplification et évaluation d'expressions

8. Eliminer la négation dans les expressions de sélection, si possible
   Exemple1 : *"not x = (select ...)"* $\longrightarrow$ *"x <> (select ...)"*
   Exemple2 : *"not (x < 123 or y is null)"* $\longrightarrow$ *"x >= 123 or y is not null"*.

2.2/5- Réécriture de requêtes complexes

- Transformations décidées en fonction de la complexité de la requête, de la présence de vue ou d'index, etc.
- Types de transformation :
  1. Requête simple $\longrightarrow$ Requête composée,
  2. Désimbriquer des sous-requêtes,
  3. Incorporer des définitions de vues dans une requête
  4. Introduire des prédicats de la requête dans une vue.

2.2/5- Requête simple $\longrightarrow$ Requête composée

select ... where A = Constante1 or B = Constante2

réécrite en :

(select ... where A = Constante1)
union (select ... where B = Constante2)

Si index sur A et B.

2.2/5- Désimbriquer des sous-requêtes

- Traiter la requête sous sa forme initiale
- ou désimbriquer la sous-requête en introduisant des jointures dans la requête initiale.
- Exemple : Si contrainte de clé primaire ou d'unicité sur *R2.x*

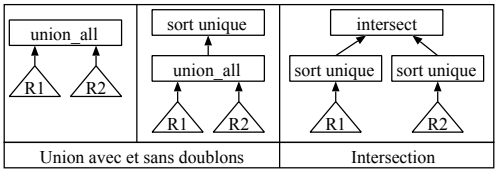
  select ... from R1
  where x in (select x from R2 where condition) ;

  réécrite en :

  select ... from R1, R2
  where condition and R1.x = R2.x.

2.2/5- Cas ded Requêtes composées

- Cas requête avec opérateurs ensemblistes ($\cup, \cap, \setminus$) :
  1. Un plan d'exécution pour chaque membre de l'opérateur
  2. Composition du résultat avec l'opérateur ensembliste
- Exemples : (triangle $\simeq$ sous-arbre)



2.3/5- Traitement des requêtes avec vues

(a) Intégrer la définition de la vue à la requête

(b) Etendre la définition de la vue par des prédicats de la requête.

2.3.a/5- Intégration de la vue à la requête

- Faisable sur les *vues* dites *intégrables* ou *fusionnables* ne comportant pas, dans la liste de projection, :
  - d'opérations ensemblistes
  - de clause *connect*,
  - de pseudo-colonne *rownum*,
  - de fonction d'agrégation (*min, avg, main, max, sum*)
- Exemple :

  "select x1 from V where Condition2"

  et *"create view V as (select x1, . . .from R where Condition1)"*

  $\longrightarrow$ *"select x1 from R where Condition1 and Condition2"*

2.3.b/5- Intégration de prédicats de la requête à la vue

- Quand échec de l'intégration de la vue à la requête

select x1, x2 *from* V
where xk = 1234
avec V définie par :
create view V(x1, ..., xn) *as*
select x1, ..., xn *from R1*
union (*select* x1, ..., xn *from R2)*

est ré-écrit en

select x1, x2
from select x1, ..., xn *from R1*
where xk = 1234
union (select x1, x2
from select x1, ..., xn *from R1*
where xk = 1234)

2/5. Les étapes de l'optimisation : Plan

1. Simplification et Evaluation d'expressions (p. 60)
2. Réécriture de requêtes complexes (p. 63)
3. Réécriture de requêtes avec vues (p. 67)
4. Choix de la stratégie d'optimisation (p. 71)
5. Traitement des jointures (p. 77)

2.4/5- Stratégie et but de l'optimisation

- Stratégie (rappel) : Base de coûts/de règles
- But de l'optimisation : globale ou temps de réponse
- *Optimisation globale* : minimiser la quantité totale de ressources nécessaires au traitement de tous les tuples concernés par une instruction
  - But par défaut
  - Adaptée pour les traitements en mode différé
- *Optimisation du temps de réponse* : minimiser les ressources pour accéder au premier tuple concerné par l'instruction.
  - Adaptée pour les applications conversationnelles

2.4/5- Stratégie et but de l'optimisation : Paramétrisation

1. Niveau instance Oracle : paramètre d'initialisation *optimizer_mode*
2. Niveau session : argument *optimizer_goal* de *alter session*
3. Niveau instruction : directives d'optimisation *optimizer_mode* et *optimizer_goal*

- Si paramères utilisés comme directive d'optimisation : portée = l'instruction qui les contient.
- Si *optimizer_goal* spécifié pendant une session : substitution à la valeur de *optimizer_mode* pour la durée de la session.

2.4/5- Paramétrisation : Valeurs de *optimizer_mode* et *optimizer_goal*

.

2.4/5- Paramétrisation : Valeurs de *optimizer_mode* et *optimizer_goal*

- *choose, rule, all_rows, first_rows*

1. *choose* et présence de statistiques sur au moins une table
   - Stratégie : à base de coûts
   - But : optimisation globale
2. *choose* et absence de statistiques : stratégie à base de règles.
3. *rule :*
   - = Valeur par défaut
   - Stratégie : à base de règles (avec ou sans statistiques)
   - But : optimisation globale

2.4/5- Paramétrisation : Valeurs de *optimizer_mode* et *optimizer_goal*

4. *all_rows :* même en l'absence de statistiques,
   - Stratégie : à base de coûts
   - But : optimisation globale
5. *first_rows :*
   - Stratégie : à base de règles
   - But : minimisation du temps de réponse

2/5. Les étapes de l'optimisation : Plan

2.5/5- Traitement des jointures

- Combinaison des facteurs :
  - **(a)** Type d'algorithme de jointure
  - **(b)** Ordre des jointures
  - **(c)** Chemins d'accès
    - Dans l'optimisation à base de coûts : page 86
    - Dans l'optimisation à base de règles : page 105

2.5.a/5- Choix de l'algorithme de jointure

- Méthodes ou algorithmes de jointure :
  1. Boucles imbriquées (*nested loops*),
  2. Tri-fusion (*sort-merge*),
  3. Hachage (*hash join*),
  4. Groupement (*cluster join*).
- Critères de choix : cf. page 79 à page 81

2.5.a/5- Choix de l'algorithme de jointure

1. Boucles imbriquées :
   - Utilisable par les deux stratégies d'optimisation
   - Choisi si taille du résultat $>$ 10000 tuples
   - Coût = $C\_R\_ext + (C\_R\_int * Card(R\_ext))$
   - *C_R_ext* : coût de l'accès à la relation externe
   - *C_R_int* : coût de l'accès à la relation interne

2.5.a/5- Choix de l'algorithme de jointure

2. Tri fusion :
   - Utilisable par les deux stratégies d'optimisation
   - Le plus efficace si
     (i) optimisation à base de règles
     (ii) résultat de grande taille
   - Coût : Coûts d'accès aux relations [+ Coûts de leur tri]

2.5.a/5- Choix de l'algorithme de jointure

3. Hachage :
   - Tables partitionnées
   - Non utilisable dans une optimisation à base de règles
   - Le plus efficace si résultat de grande taille
   - Coût : $C\_R2 + (C\_R1 * Nb\_Partition\_R2)$,
     - *C_R1, C_R2* : coûts des accès aux relations
     - *Nb_Partition_R2* : nombre de partitions de hachage de *R2*

4. Groupement : utilisable dans une équi-jointure :
   - de tables d'un même *cluster*
   - sur la clé du cluster

2.5.b/5- Ordre des jointures

- Quelle que soit la stratégie d'optimisation :
  1. En tête de l'ordre : Table dont la jointure rend un seul tuple
  2. Poursuite de la recherche de l'ordre (variable selon les stratégies)
- cf. détails page 83 à page 84

2.5.b/5- Ordre des jointures et *Optimisation à base de coûts*

- Engendrer et évaluer le coût des plans en prenant en compte :
  - les chemins d'accès disponibles
  - l'ordre des jointures
  - le type d'algorithme de jointure.

2.5.b/5- Ordre des jointures et *Optimisation à base de règles*

- Engendrer un ensemble de plans d'exécution des jointures
- En choisir un en se laissant guider par un objectif :
  1. de maximisation du nombre de jointures par boucles imbriquées
  2. où la table interne est accessible *via* un index (*index scan*).

Traitement des requêtes sous Oracle : Plan

1. Architecture du compilateur de requêtes et structure de stockage des plans d'exécution (p. 50)
2. Processus et principales étapes de traitement des requêtes
3. Optimisation à base de coûts (p. 86)
4. Optimisation à base de règles (p. 105)
5. Directives d'optimisation (p. 108)

3/5. Optimisation à base de coûts

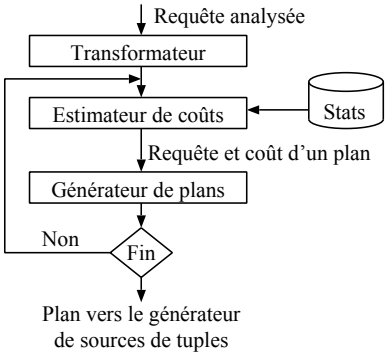
- Conditions de mise en œuvre :
  1. Disponibilité de statistiques
  2. Paramètres d'initialisation positionnés :
     (a) *optimiser_mode = choose, first_rows* ou *all_rows*
     (b) *optimiser_features_enable* = . . .
     (c) *compatible* = . . .
- *Note* : Consulter les paramètres en vigueur dans *v$parameter*
- Exemple :

```
select name, value from v$parameter
where upper(name) like upper('%optimizer%')
```


3/5. Optimisation à base de coûts

- Stratégie généralement requise pour les instructions qui utilisent :
  - des tables partitionnées et/ou organisées en index,
  - des index basés sur des fonctions.
- Processus d'optimisation :
  1. Engendrer un ensemble de plans : se fonder sur
     - les chemins d'accès existants
     - les éventuelles directives d'optimisation
  2. Estimer le coût de chaque plan : utiliser les statistiques sur
     - la distribution des données
     - les caractéristiques de stockage des objets (tables, index, partitions) impliqués
  3. Choisir le plan le moins coûteux.

3/5. Optimisation à base de coûts : Architecture



Collecte de statistiques

- Calculer/estimer (*compute/estimate statistics*) des statistiques sur les espaces ;
- *Calcul* sur la totalité de l'instance d'un objet (table, index, partition ou cluster) ;
- *Estimation* : sur un échantillon spécifié en
  - nombre de lignes (1064 par défaut)
  - ou en pourcentage de la taille de l'objet.
- Clauses de portée de la commande d'analyse (*analyze*) :
  - *for table* : informations sur la table,
  - *for columns* : une liste de ses colonnes

Collecte de statistiques : ExempleS

Collecte de statistiques : (1/2) Sur des tables

Nom de colonne	Nature de l'information
NUM_ROWS	Nombre de tuples
BLOCKS	Nombre de blocs de données
EMPTY_BLOCKS	Nombre de blocs qui n'ont jamais été occupés
AVG_SPACE	Taux moyen d'espace libre par bloc
CHAIN_COUNT	Nombre de tuples ayant subi une migration
AVG_ROW_LEN	Taille moyenne d'un tuple

Collecte de statistiques : (2/2) Sur des Index

Nom de colonne	Nature de l'information
BLEVEL	Nombre de niveaux d'index
LEAF_BLOCKS	Nombre de blocs feuilles
DISTINCT_KEYS	Nombre de valeurs distinctes de la clé de l'index
AVG_LEAF_BLOCKS_PER_KEY	Nombre moyen de blocs feuilles par valeur de la clé de l'index
AVG_DATA_BLOCKS_PER_KEY	Idem pour les blocs de données
CLUSTERING_Factor	Facteur de groupement des clés de l'index

Statistiques et tables du dictionnaire

- Stats sur tables : Voir *user_tables, all_tables* et *dba_tables*
- Sur colonnes : {*user_* | *all_* | *dba_*}*tab_columns*
- Sur clusters : (en partie) vues {*all_* | *user_* | *dba_*}*clusters*.

3/5. Optimisation à base de coûts

- La suite :
  1. Chemins d'accès (p. 95)
  2. Transformation de requêtes (p. 98)
  3. Estimation de coûts (p. 100)
  4. Génération de plans (p. 103)

3.1/5. Optimisation à base de coûts : Chemins d'accès

- (Terminologie d'Oracle) Chemin d'accès (*access path*) :
  1. Mode d'accès
  2. Mode de parcours éventuel des tuples d'un objet.
- Quelques types de chemins d'accès :
  - *Full table scans* : parcours de toute la table ;
  - *Sample table scans* : parcours d'un échantillon de tuples ;
  - *Table access by RowId* : accès à l'aide d'un identifiant de tuple ;
  - *Index scans* : mode de parcours d'index avec des variantes dont :
    - *unique scan* : lorsqu'un identifiant de tuple est rendu ;
    - *range scan* : étant donné un intervalle de valeurs de la clé ;
    - *full scan* : parcours de la totalité d'un index ;
    - *fast full scan* : l'index suffit pour satisfaire la requête.

3.1/5. Optimisation à base de coûts : Chemins d'accès

- Des facteurs influant sur le *choix d'un chemin d'accès* :
  - Liste des chemins disponibles,
  - Estimation du gain potentiel en utilisant tout ou partie des chemins,
  - *Sélectivité* des attributs et de la jointure,
  - Nombre de valeurs de chaque colonne d'une table (dans *user_tab_columns_num_distinct*),
  - Nombre de tuples d'une table (*user_tab_columns_num_rows*),
  - Statistiques disponibles (*user_tab_col_statistics* et *user_tab_columns*).

3/5. Optimisation à base de coûts

- La suite :
  1. Chemins d'accès (p. 95)
  2. Transformation de requêtes (p. 98)
  3. Estimation de coûts (p. 100)
  4. Génération de plans (p. 103)

3.2/5- Optimisation à base de coûts : Transformation de requêtes

- Objectif du transformateur de requêtes : est-il avantageux d'opérer des réécritures de requêtes ?
- Evaluation de l'opportunité
  1. d'intégrer les définitions de vues à la requête,
  2. de désimbriquer des requêtes,
  3. d'utiliser des *vues matérialisées*.
- **Note** : *Vue matérialisée* = Vue instanciée ($\simeq$ Table)

3.2/5- Optimisation à base de coûts : Transformation de requêtes

1. Intégration des vues
   - Cas vue *fusionnable* avec l'instruction : Génération d'un seul plan
   - Cas vue *non fusionnable* : Génération d'un sous-plan pour la vue
2. Sous-requêtes :
   - Certaines peuvent être désimbriquées et d'autres pas
   - Celles non désimbriquées $\longrightarrow$ Plans séparés + Ordre entre les plans.
3. Vues matérialisées : si une partie de la requête correspond à la définition d'une vue matérialisée, la remplacer par le nom de la vue.

3.3/5- Estimation des coûts

- Facteurs :
  1. *Sélectivité*,
  2. Cardinal des ensembles
  3. Coût des ressources utilisées : principale unité = nombre d'entrées-sorties physiques (blocs de données et d'index).
- *Sélectivité*, sans disponibilité de statistiques :
  - Utilisation d'une valeur interne est utilisée
  - Exemple : Sélectivité d'une d'une expression du type *"attribut = valeur"* est estimée meilleure que celle d'une expression du type *"attribut > valeur"*.

3.3/5- Optimisation à base de coûts : Estimations

- *Sélectivité* et présence de statistiques :
  - Calcul de la sélectivité
  - Exemple : Sélectivité d'une expression *"attribut = valeur"* = (1/nombre de valeurs distinctes de l'attribut).
- *Cardinal des ensembles* :
  - Tables, Vues,
  - Résultats de jointures et de requêtes avec *group by*
  - Calcul d'un *cardinal effectif* en utilisant la sélectivité
  - Si défaut de statistiques : Estimation d'un *cardinal de base* en fonction du nombre d'*extents* occupés.

3/5. Optimisation à base de coûts

- La suite :
  1. Chemins d'accès (p. 95)
  2. Transformation de requêtes (p. 98)
  3. Estimation des coûts (p. 100)
  4. Génération de plans (p. 103)

3.4/5- Optimisation à base de coûts : Génération de plans

- Un sous-plan :
  - pour chaque sous-requête désimbriquée
  - pour chaque vue non intégrée à la requête.
- Génération et optimisation du plan global de bas en haut (du bloc de requête le plus interne vers la requête)
- Arrêt lorsque l'évaluateur estime que le coût du dernier plan engendré est "acceptable"
  (i.e. rapport gain escompté-coût examen exhaustif)

Traitement des requêtes sous Oracle : Plan

1. Architecture du compilateur de requêtes et structure de stockage des plans d'exécution (p. 50)
2. Processus et principales étapes de traitement des requêtes
3. Optimisation à base de coûts (p. 86)
4. Optimisation à base de règles (p. 105)
5. Directives d'optimisation (p. 108)

4/5. Optimisation à base de règles

- Choix du plan d'exécution en fonction
  1. des chemins d'accès existants
  2. d'un poids associé à chaque type de chemins,
- Meilleur chemin : celui de plus faible poids
- Quinze types de chemins disponibles
- Leur choix dépend :
  1. de la forme de la requête
  2. de la structure des objets concernés

4/5. Optimisation à base de règles : Exemples de choix de chemins d'accès

Chemin d'accès	**Conditions et forme de la requête**
Single Row by RowId	*where rowid = '...'*
Single Row by cluster join	*where* R1.A = R2.A *and* R1.B = valeur et R1, R2 groupées (*cluster*) sur A et B est clé primaire de R1.
single row by unique or primary key	La clause *where* porte sur tous les attributs d'une clé primaire ou d'un index *unique*.
bounded range search or index columns	*where* A = expression ou *where* A > [=] expression1 *and* A > [=] expression2.

Traitement des requêtes sous Oracle : Plan

1. Architecture du compilateur de requêtes et structure de stockage des plans d'exécution (p. 50)
2. Processus et principales étapes de traitement des requêtes
3. Optimisation à base de coûts (p. 86)
4. Optimisation à base de règles (p. 105)
5. Directives d'optimisation (p. 108)

5/5. Directives d'optimisation

- Pour forcer certains choix de l'optimiseur
- Sous la forme : "... /* + *directives* */ ..." dans une instruction
- Exemples :

5/5. Directives d'optimisation et leurs objets

1. Méthode et but de l'optimiseur : *all_rows*, *first_rows*, *choose*, *rule*.
   - *all_rows*, *first_rows*, *choose* $\longrightarrow$ invoquer l'optimiseur à base de coûts
2. Méthode d'accès : *full, rowid, index, cluster, index_join, rewrite,* etc.
3. Ordre des jointures : *ordered*.
4. Algorithme de jointure :
   - *use_nl* : boucles imbriquées,
   - *use_merge* : tri-fusion),
   - *use_hash* : fonction de hachage
   - *use_nl*, *use_merge* recommandés conjointement avec *ordered*

5/5. Directives d'optimisation et leurs objets

5. Fusion (ou pas) de vues :
   - *merge, no_merge*,
   - *push_pred*, *no_push_pred*.
6. Désimbrication (ou non) de sous-requêtes : *unnest, no_unnest*.
7. Stratégie de gestion en mémoire cache : *cache, nocache*.
8. Exécution parallèle d'instructions : *parallel, noparallel, parallel_index* et *noparallel_index*.
9. etc.

Traitement des requêtes : Conclusion

- Optimisation algébrique : Fondement = Propriété des opérateurs relationnels
- Algèbre $\simeq$ Langage de base (”Assembleur”) des SGBDR
- Optimisation à base de coûts : disponibilité de statistiques
- Concepts et techniques à connaître pour :
  - une vision non naïve des SGBD
  - amélioration des performances
  - administration des applications (et des systèmes)