Optimisation de requêtes

Illustration avec Oracle

J. Akoka - I. Wattiau

Introduction

BUT :

Fournir l'algorithme d'accès à la base de données pour répondre à une requête exprimée en langage assertionnel

LES ETAPES :

1- Analyse de la requête

↙ **correction et simplification**

2- Ordonnancement en séquence d'opérations élémentaires

↙ **plan d'exécution**

3- Exécution

L'optimisation des requêtes

consiste à :

- **maximiser le parallélisme entre les entrées-sorties**
- **minimiser :**
  - **le nombre d'E/S**
  - **la taille mémoire nécessaire à l'exécution**
  - **le temps unité centrale**

L'analyse de la requête (1)

Sur un exemple :

COURS (code, nomcours, nomprof)

NOTE (code, matricule, note)

ETUDIANT (matricule, nom, adresse)

Question :

Quels sont les noms des cours suivis par l'étudiant

DUPONT donnés par le professeur DURAND ?

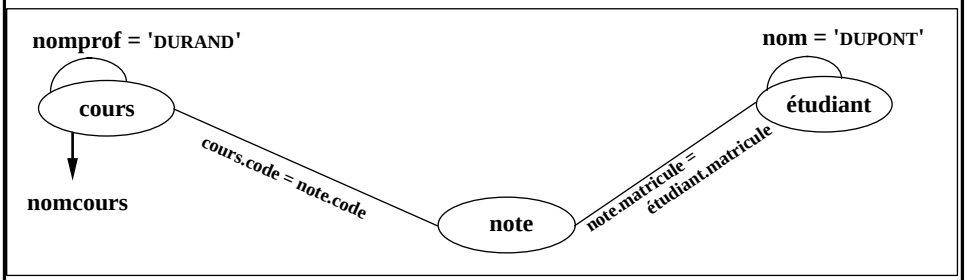
L'analyse de la requête (2)

En SQL :

```
Select nomcours
From cours
Where nomprof = 'DURAND'
      and code in (select code from note
where matricule in (select matricule
from etudiant where nom = 'DUPONT'));
```

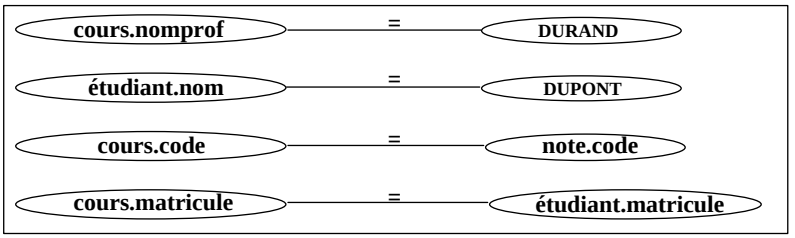
Outils de représentation des requêtes (1)

GRAPHE DE CONNEXION DES RELATIONS



Outils de représentation des requêtes (2)

GRAPHE DE CONNEXION DES ATTRIBUTS



- Condition nécessaire pour que la question soit correcte : connexité du graphe des relations et qualifications cohérentes
- 2 requêtes sont équivalentes si elles ont des graphes d'attributs transitivement identiques
- On peut compléter les graphes en y représentant les contraintes d'intégrité, on peut générer ainsi d'autres requêtes équivalentes

Ordonnancement de la requête

L'analyse de la requête permet de déterminer l'ensemble des opérations de l'algèbre relationnelle à exécuter :

- **une opération peut être exécutée dès que ses opérandes sont disponibles**
- **si l'opération A n'utilise pas le résultat de l'opération B et vice-versa, A et B peuvent être exécutées en parallèle**

Identification des problèmes de performance

- Non existence d 'index appropriés
- non re-compilation de procédures stockées après modifications importantes de la base
- existence de statistiques obsolètes

Décision d'optimisation

- Quand une requête s 'exécute plus lentement que des requêtes similaires
- quand le temps d 'exécution est jugé anormal par rapport à la taille des tables et par rapport aux index existants
- quand un plan n 'utilise pas un index qu'il devrait
- quand le temps d 'exécution d 'une requête dans une procédure stockée est supérieur à celui de la requête directe

Processus d ’optimisation

- Dans les SGBD du marché, combinaison de deux méthodes
  - méthode dite syntaxique
    - principalement restructurations algébriques
  - méthode d’estimation de coûts

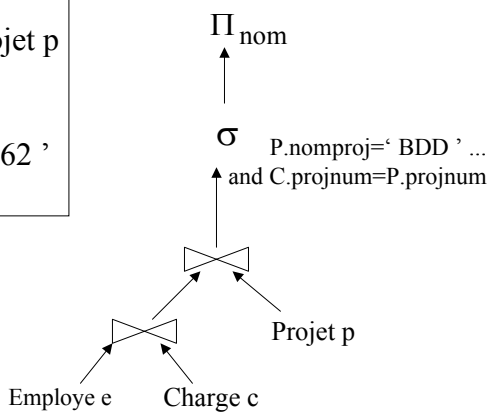
Optimisation par restructurations algébriques

0. Ecrire la suite des opérations
1. Découper les restrictions comportant plusieurs prédicats
2. Remonter les restrictions le plus haut possible
3. Regrouper les restrictions successives portant sur une même relation
4. Refaire 2 et 3 pour les projections

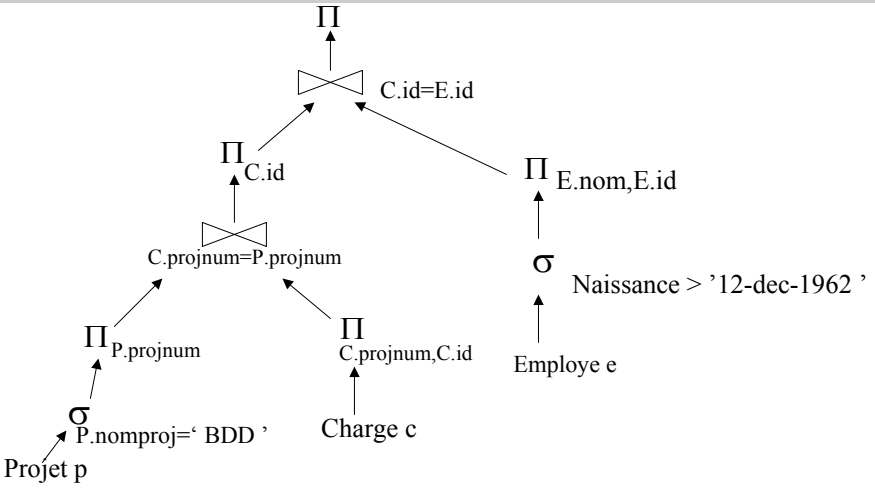
Arbre algébrique

```
Select e.nom
from charge c, employe e, projet p
where p.nomproj=‘ BDD’
and e.id=c.id
and e.naissance > ‘ 31-dec-1962 ’
and c.projnum=p.projnum;
```

Hypothèses :
projet 20 tuples de 100 octets
charge 100 tuples de 50 octets
employé 500 tuples de 100 octets



Arbre algébrique restructuré



Optimisation par estimation de coûts

- Évaluer le coût d ’exécution de la requête en utilisant des statistiques sur :
  - nb exact ou estimé de tuples et de blocs par table
  - nb de niveaux d’index et nb de valeurs distinctes par attribut :
    - sélectivité d ’un attribut = nb de val. différentes / nb tuples (1 pour les clés)
  - nb estimé de L/E logiques et physiques, ...

L'optimiseur Oracle

Analyse de la requête

- vérification de la syntaxe
- recherche des tables et des colonnes dans le dictionnaire de données

Ordonnancement en séquence d'opérations élémentaires

- génération d'un méta-code, requête SQL combinée avec des références à des index, des réservations d'espaces, etc ...

L 'optimiseur Oracle

Exécution

- recherche dans les index
- si nécessaire, transfert de blocs en mémoire centrale
- calcul de la réponse
- affichage de la réponse

Optimisation dans Oracle

- Deux stratégies d 'optimisation :
  - à base de règles (syntaxique)
  - à base de coûts
- On choisit pour une requête, pour une session ou pour une instance
- On peut aussi incorporer des directives d 'optimisation dans une requête

Utilisation des index

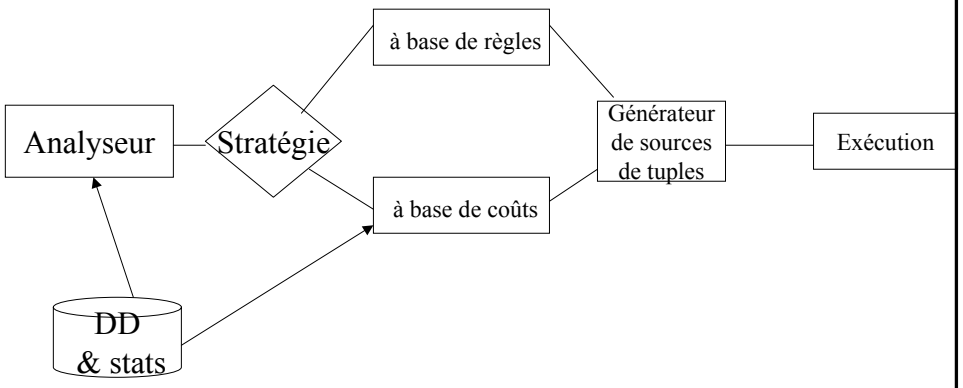
dans ORACLE, les index sont utilisés selon la forme des clauses WHERE selon l'ordre suivant :

1 ROWID = cste
2 Colonne d'index Unique = cste
3 Index concaténé unique = cste
4 Index concaténé non unique = cste
5 Colonne d'index non unique = cste
6 Index concaténé = cste

Utilisation des index

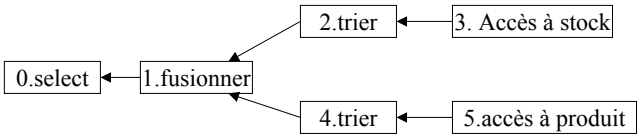
7 Partie gauche d'un index concaténé = cste
8 Colonne d'index unique BETWEEN cste
9 Colonne d'index non unique BETWEEN cste AND cste ou LIKE 'cste%'
10 Colonne d'index unique > cste
11 Colonne d'index non unique > cste ou < cste
12 SORT/MERGE (jointures seulement)
13 MAX (colonne indexée) ou MIN
14 ORDER BY index AND cste ou LIKE"cste%"
15 Parcours complet de la table

Architecture de l 'optimiseur



Exemple de plan d 'exécution

Select p.libelle, s.qte from produit p, stock s where p.prodnum=s.prodnum;



Explain plan set statement_id=' XXX ' for select p.libelle ….

Puis select id, parent_id, operation, object_base,options from plan_table;

Id	parent_id	operation	object_base	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

Exemples d ’opérations

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 jointures externes
merge join		jointure par tri fusion
	outer	idem pour jointures externes
	semi	idem pour semi-jointure
view		interrogation d'une vue

Exemple

Cinéma(id-cinéma*, nom, adresse)
salle(id-salle*, nom, capacité^,id-cinéma)**
film(id-film*,titre, année, idréalisateur)**
séance(id-séance*,heuredébut,heurefin,idsalle,idfilm)**
artiste(id-artiste*,nom,datenaissance)
*** index unique**
**** index non unique**
^ avec ou sans

SELECTION SANS INDEX

- SELECT * FROM cinema WHERE nom='Le Rex';
- Plan d ’exécution :
  - 0 SELECT STATEMENT
  - 1 TABLE ACCESS FULL CINEMA
    L'opération 1 est très coûteuse car elle balaie entièrement la table Cinéma

SELECTION AVEC INDEX

- SELECT * FROM cinema WHERE ID-cinéma=1908;
- Plan d'exécution :
  - 0 SELECT STATEMENT
  - 1 TABLE ACCESS BY INDEX ROWID CINEMA
  - 2 INDEX UNIQUE SCAN IDX-CINEMA-ID
- Avec ID-Cinema : index unique

SELECTION CONJONCTIVE AVEC INDEX

- SELECT capacité FROM salle WHERE ID-cinéma=187 AND nom='Salle 1';
- Plan d'exécution :
  - 0 SELECT STATEMENT
  - 1 TABLE ACCESS BY INDEX ROWID SALLE
  - 2 INDEX RANGE SCAN IDX-SALLE-CINEMA-ID
- Avec ID-Cinema : index non unique

SELECTION CONJONCTIVE AVEC 2 INDEX

- SELECT nom FROM salle WHERE ID-cinema=1098 AND capacité=150
- Plan d'exécution :
  - 0 SELECT STATEMENT
  - 1 TABLE ACCESS BY ROWID SALLE
  - 2 AND-EQUAL
  - 3 INDEX RANGE SCAN IDX-SALLE-CINEMA-ID
    4 INDEX RANGE SCAN IDX-CAPACITE
- Avec ID-cinema et Capacité : index non unique (sans index = id précédente)

SELECTION DISJONCTIVE AVEC DES INDEX

- SELECT nom FROM salle WHERE ID-cinema=1098 OR capacité>150
- Plan d'exécution :
  - 0 SELECT STATEMENT
  - 1 CONCATENATION
  - 2 TABLE ACCESS BY ROWID SALLE
  - 3 INDEX RANGE SCAN IDX-CAPACITE
  - 4 TABLE ACCESS BY ROWID SALLE
    5 INDEX RANGE SCAN IDX-SALLE-CINEMA-ID
- Avec ID-cinema et Capacité : index non unique

SELECTION AVEC ET SANS INDEX

- SELECT nom FROM salle WHERE ID-cinema=1098 OR nom='salle 1'
- Plan d'exécution :
  - 0 SELECT STATEMENT
  - 1 TABLE ACCESS FULL SALLE
- Avec ID-cinema : index non unique
- l 'index n 'est pas utilisé

JOINTURE AVEC 1 INDEX

- SELECT film.*FROM film, séance WHERE Film.idfilm=séance.idfilm;
- Plan d'exécution :
  - 0 SELECT STATEMENT
  - 1 NESTED LOOPS
  - 2 TABLE ACCESS FULL séance
  - 3 TABLE ACCESS BY ROWID FILM
  - 4 INDEX UNIQUE SCAN IDFILM_IDX
- Avec film.idfilm : index unique

JOINTURES AVEC 2 INDEX

- SELECT cinema.nom, capacite FROM cinema, salle WHERE cinema.ID-cinema=salle.ID-cinema;
- Plan d'exécution :
  - 0 SELECT STATEMENT
  - 1 NESTED LOOPS
  - 2 TABLE ACCESS FULL SALLE
  - 3 TABLE ACCESS BY ROWID CINEMA
  - 4 INDEX UNIQUE SCAN IDX-CINEMA-ID
- Avec cinema.ID-cinema : index unique et salle.ID-cinema : index non unique

JOINTURE AVEC 2 INDEX ET SELECTION AVEC 1 INDEX

- SELECT cinema.nom, capacite FROM cinema, salle WHERE cinema.ID-cinema=salle.ID-cinema AND capacite>150;
- Plan d'execution :
- 0 SELECT STATEMENT
  - 1 NESTED LOOPS
  - 2 TABLE ACCESS BY INDEX ROWID SALLE
  - 3 INDEX RANGE SCAN IDX-CAPACITE
  - 4 TABLE ACCESS BY INDEX ROWID CINEMA
  - 5 INDEX UNIQUE SCAN IDX-CINEMA-ID
- Avec cinema.ID-cinema : index unique et salle.ID-cinema : index non unique et capacite : index non unique

Jointures par boucles imbriquées

- Parcourir les tuples d ’une table (relation externe)
- pour chaque tuple sélectionné, trouver les tuples dans l ’autre table (relation interne) qui joignent
- dans la relation externe, les tuples sont accédés une seule fois
- dans la relation interne, sans index, les tuples sont accédés de nombreuses fois
- => importance du choix de l’ordre des jointures

Traitement des jointures

- 4 algorithmes
  - boucles imbriquées -> si tables de grande taille
  - tri-fusion
  - hachage
  - par groupement -> en cas de cluster

Directives d 'optimisation

- **Choose :** stratégie à base de coûts avec optimisation globale si des stats existent, sinon à base de règles
- **Rule :** stratégie à base de règles (défaut)
- **all_rows :** à base de coûts avec optimisation globale
- **first_rows :** à base de règles avec minimisation du temps de réponse