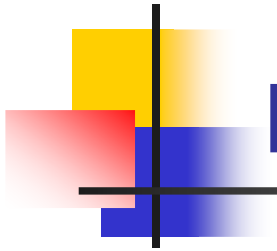
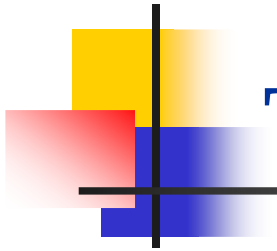
Chapitre 2

Évaluation et optimisation de requêtes



PLAN

1. Analyse d'une requête
   - Étapes
   - Décomposition
   - Traduction algébrique et plan d'exécution
2. Optimisation des requêtes
   - Principes
   - Optimisation logique: Règles de réécriture
   - Optimisation physique
   - Algorithmes de jointure: boucles imbriquées, tri fusion, index



Traitement d'une requête

Requêtes exprimées en SQL: *langage déclaratif* – On indique *ce que l'on veut* obtenir, on ne dit pas *comment* l'obtenir

Le SGBD doit faire le reste

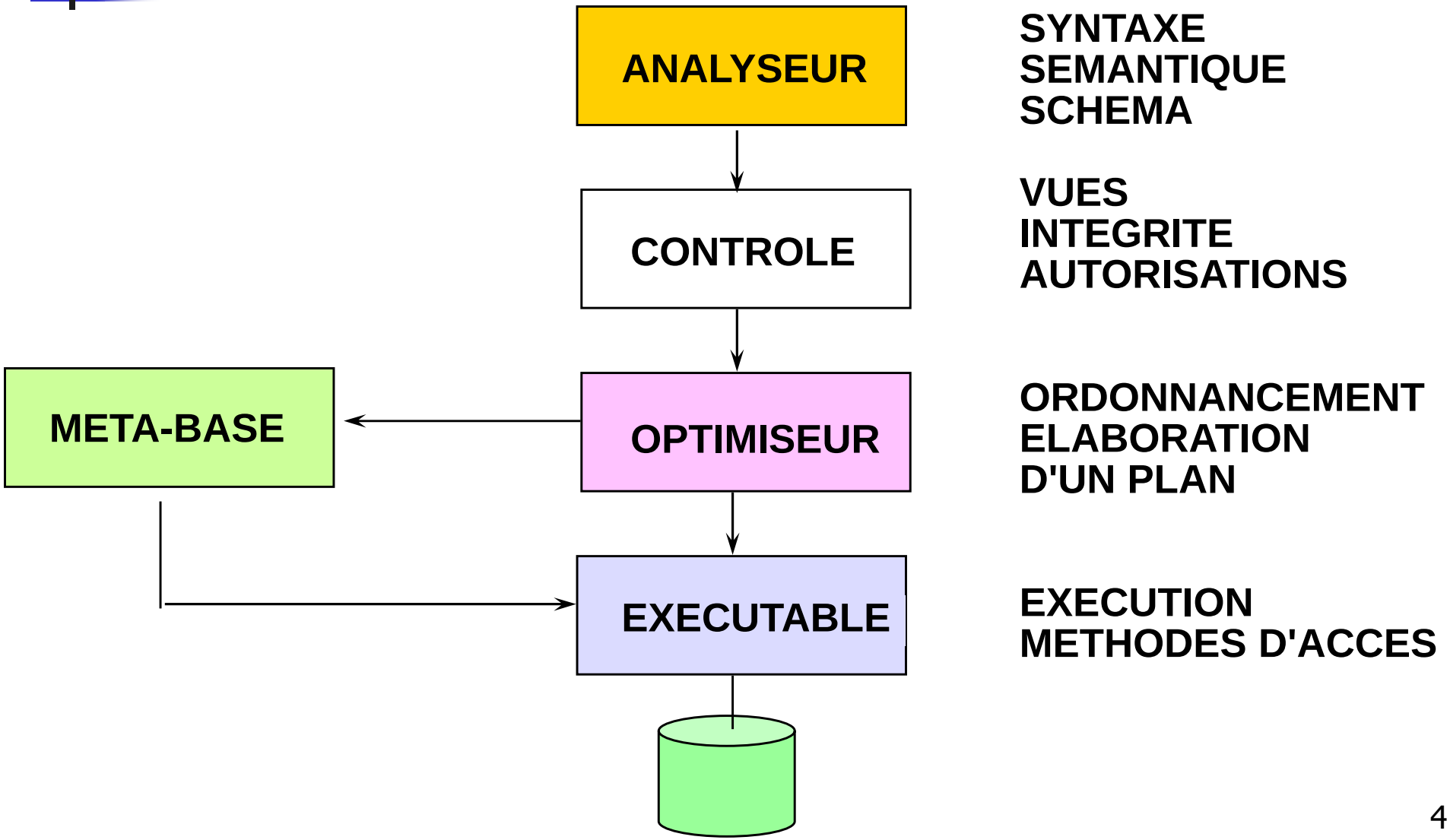
- Déterminer la façon d'exécuter la requête: *plan d'exécution*
- Plusieurs plans possibles, choisir le meilleur : *optimisation* –

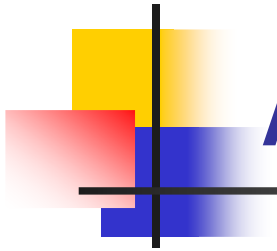
Exécuter le plan choisi: *évaluation*

Plan d'exécution

- Exprimé en *algèbre* relationnelle (expression algébrique)
- Forme *exécutable*: on sait précisément *comment* l'évaluer

1. ARCHITECTURE TYPE SGBD





Analyse de la requête

- Sous-étapes
  - Analyse syntaxique
  - Analyse sémantique
  - Simplification – Normalisation traduction algébrique

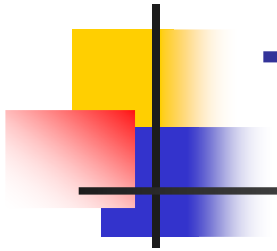
- Analyse syntaxique
  - Requête = chaîne de caractères
  - Contrôle de la structure grammaticale (respect de la syntaxe SQL)
  - Vérification de l'existence des relations/attributs adressés dans la requête (Utilisation du dictionnaire de données de la base)

Analyse de la requête

- Analyse sémantique
  - Vérification des opérations réalisés sur les attributs *Ex*. Pas d'addition sur un attribut texte
  - Détection d'incohérences, *Ex*. *prix*=5 *and prix*=6

- Simplification
  - Conditions inutilement complexes, *Ex*. *(A or not B) and B* est équivalent à *A and B*

- Normalisation: simplifie la traduction algébrique
  - Transformation des conditions en forme normale conjonctive
  - Décomposition en blocs *Select-From- Where*



Traduction algébrique

- Déterminer l'expression algébrique équivalente à la requête SQL
- Clause SELECT - opérateur de projection
- Clause FROM - les relations qui apparaissent dans l'expression
- Clause WHERE

- Condition "*Attr* = *constante*" opérateur de sélection
- Condition "*Attr1* = *Attr2* jointure ou sélection

- Résultat: expression algébrique – Représentée par un *arbre de requête*

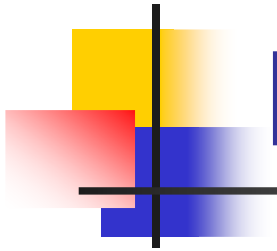
Exemple

Cinéma (*ID-cinéma*, nom, adresse) – ***Salle*** (*ID-salle*, ID-cinéma, capacité) ***Séance*** (*ID-salle*, *heure-début*, film)

- Requête: quels films commencent au Multiplex à 20 heures?

SELECT Séance.film
FROM Cinéma, Salle, Séance
WHERE Cinéma.nom = 'Multiplex' AND Séance.heure-début = 20 AND Cinéma.ID-cinéma = Salle.ID-cinéma AND Salle.ID-salle = Séance.ID-salle

- Expression algébrique
  - $\pi_{film}\,(\sigma_{nom\ =\ \text{'Multiplex'}\ \wedge\ heure\text{-}début\ =\ 20}\,((\textbf{\textit{Cinéma}} \infty \textbf{\textit{Salle}}) \infty \textbf{\textit{Séance}})$



ETAPES DE L'OPTIMISATION

(1) Obtention d’une représentation canonique

(2) Réécriture = transformation par :

- simplification
- ordonnancement des opérations élémentaires

(3) Planning = construction des plans d'exécution candidats

choix des algorithmes pour chaque opérateur,

calcul du coût de chaque plan,

choix du meilleur plan.

Etapes 1 et 2 : indépendantes des données

Etape 3 : dépendante des données

RESTRUCTURATION ALGEBRIQUE

- Problème :
- suivant l'ordre des opérateurs algébriques dans un arbre, le coût d'exécution est diffèrent

- Pourquoi?
- 1. le coût des opérateurs varient en fonction du volume des données traitées

  i.e., plus le nombre de tuples des relations traitées est petit, plus les coûts cpu et d'E/S sont minimisés
- 2. certains opérateurs diminuent le volume des données

  e.g., restriction et projection

RESTRUCTURATION ALGEBRIQUE

- Pour une requête SQL, il y a plusieurs expressions algébriques *équivalentes* possibles

- On passe d'une expression à une autre équivalente en utilisant des **règles de réécriture**

- Le rôle de l'optimiseur:
  - Trouver l'expression algébrique optimale (temps d'exécution minimal) en appliquant les règles de réécriture

Les règles de réécriture

- Soient E_1, E_2, E_3 des expressions relationnelles, soient F_1 et F_2 des conditions et $\otimes$ l'opérateur de jointure

- La commutativité :

$$E_1 \otimes_{F_1} E_2 \Leftrightarrow E_2 \otimes_{F_1} E_1 \qquad E_1 \times E_2 \Leftrightarrow E_2 \times E_1$$

$$E_1 \otimes E_2 \Leftrightarrow E_2 \otimes E_1$$

- Associativité :

$$(E_1 \otimes E_2) \otimes E_3 \Leftrightarrow E_1 \otimes (E_2 \otimes E_3) \qquad (E_1 \times E_2) \times E_3 \Leftrightarrow E_1 \times (E_2 \times E_3)$$

Les règles de réécriture

- Règles pour les restrictions :

Règle: pousser (tant que possible) les restriction vers le fond de l'arbre algébrique (c'est-à-dire vers les feuilles de l'arbre).

1. Règle de regroupement/décomposition

$$\sigma_{F_1 \wedge F_2}(E) \Leftrightarrow \sigma_{F_1}(\sigma_{F_2}(E)) \qquad \text{(R1)}$$

$$\sigma_{F_1 \vee F_2}(E) \Leftrightarrow \sigma_{F_1}(E) \cup \sigma_{F_2}(E) \qquad \text{(R2)}$$

$$\sigma_{F_1}(\sigma_{F_2}(E)) \Leftrightarrow \sigma_{F_2}(\sigma_{F_1}(E)) \qquad \text{(R3)}$$

Les règles de réécriture

2. Distribution de la restriction par rapport au produit cartésien et à la jointure

- La restriction doit être avancée uniquement à l'opérande dont les attributs sont concernés par la condition. (on suppose que la condition F porte sur des attributs de E_1)

$$\sigma_F(E_1 \times E_2) \Leftrightarrow \sigma_F(E_1) \times E_2 \quad \text{(R4)}$$

$$\sigma_F(E_1 \otimes E_2) \Leftrightarrow \sigma_F(E_1) \otimes E_2 \quad \text{(R5)}$$

$$\sigma_F(E_1 \otimes_{F_1} E_2) \Leftrightarrow \sigma_F(E_1) \otimes_{F_1} E_2 \quad \text{(R6)}$$

- Cas ou F est une condition qui ne combine que des attributs partagés par E_1 et E_2:

$$\sigma_F(E_1 \otimes E_2) \Leftrightarrow \sigma_F(E_1) \otimes \sigma_F(E_2) \quad \text{(R7)}$$

Les règles de réécriture

3. Distribution de la restriction par rapport à l'union , l'intersection et la différence.

$$\sigma_F(E_1 \cup E_2) \Leftrightarrow \sigma_F(E_1) \cup \sigma_F(E_2) \qquad \text{(R8)}$$

$$\sigma_F(E_1 \cap E_2) \Leftrightarrow \sigma_F(E_1) \cap \sigma_F(E_2) \qquad \text{(R9)}$$

$$\sigma_F(E_1 - E_2) \Leftrightarrow \sigma_F(E_1) - \sigma_F(E_2) \qquad \text{(R10)}$$

ou $\sigma_F(E_1 - E_2) \Leftrightarrow \sigma_F(E_1) - E_2$

Les règles de réécriture

- Règles pour les projections :

- Pousser une projection revient à introduire une nouvelle projection sans changer la projection initiale.
- Règle générale: On peut introduire une projection n'importe où dans le plan d'exécution, pourvu qu'elle élimine uniquement des attributs non utilisés par les opérateurs suivants et qui ne sont pas dans le résultat final.

$$\pi_{A1,A2,\ldots,An}(E) \Leftrightarrow \pi_{A1,A2,\ldots,An}(\pi_{B1,B2,\ldots,Bm}(E)) \quad \text{(R11)}$$

(A1,A2,…,An doivent être inclus dans B1,B2,…,Bm)

Les règles de réécriture

- Inversion restriction-projection :

Si la condition F ne combine que les attributs A1, …,An, alors :

$$\pi_{A1,A2,\ldots,An}(\sigma_F(E)) \Leftrightarrow \sigma_F(\pi_{A1,A2,\ldots,An}(E)) \qquad \text{(R12)}$$

si la condition F combine les attributs B1,..,Bm qui ne sont pas parmi les A1,…,An , alors :

$$\pi_{A1,A2,\ldots,An}(\sigma_F(E)) \Leftrightarrow \pi_{A1,A2,\ldots,An}(\sigma_F(\pi_{A1,A2,\ldots,An,B1,\ldots,Bm}(E)))$$

Les règles de réécriture

- Distribution projection-produit cartésien

Si A1,…,An est une liste d'attributs dans laquelle A1,…,Am sont des attributs de E1 et les attributs restants A_{m+1},…,A_n sont de E2, alors :

$$\pi_{A1,A2,\ldots,An}(E_1 \times E_2) \Leftrightarrow \pi_{A1,\ldots,Am}(E_1) \times \pi_{Am+1,\ldots,An}(E_2)) \qquad \text{(R13)}$$

- Distribution projection-union

$$\pi_{A1,\ldots,An}(E_1 \cup E_2) \Leftrightarrow \pi_{A1,\ldots,An}(E_1) \cup \pi_{A1,\ldots,An}(E_2)) \qquad \text{(R14)}$$

Algorithme d'optimisation logique

- Principe de base: pousser (descendre) les opérateurs unaires au maximum au niveau de l'arbre.
- Algorithme:

1. Séparer les restrictions comportant plusieurs prédicats à l'aide des règles R1 et R2 appliquées de la gauche vers la droite

2. Descendre les restrictions aussi bas que possible en appliquent les règles R4 … R10

Algorithme d'optimisation logique

3. Regrouper les restrictions successives portant sur une même relation à l'aide des règles R1 et R2 appliques cette fois de droite vers la gauche

4. Descendre les projections aussi bas que possible en appliquant les règles R11 … R14

5. Regrouper les projections successives sur la même relations et éliminer d'éventuelles projections qui auraient pu apparaitre (projection sur tous les attributs d'une table)

- Remarque : L'étape 3 et l'étape 5 permettent de diminuer les accès à la relation.

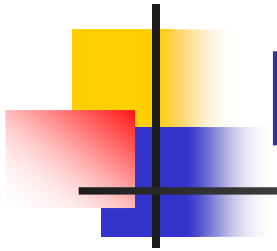
L'optimisation sur un exemple

- Considérons le schéma :

CINEMA(Cinema;Adresse;Gerant)

SALLE(Cinema;NoSalle; Capacite)

- avec les hypothèses :
  - Il y a 300 n-uplets dans CINEMA, occupant 30 pages.
  - Il y a 1200 n-uplets dans SALLE, occupant 120 pages.



L'optimisation sur un exemple

- On considère la requête : *Adresse des cinémas ayant des salles de plus de 150 places*
- En SQL, cette requête s'exprime de la manière suivante :

```
SELECT Adresse
FROM CINEMA, SALLE
WHERE capacité > 150
AND CINEMA.cinéma = Salle.cinéma
```

L'optimisation sur un exemple

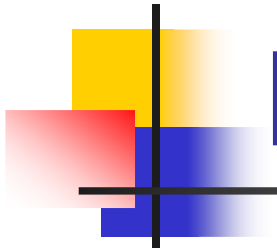
- Traduit en algèbre, on a plusieurs possibilités. En voici deux :

$$\pi_{Cinema}\,(\sigma_{Capacité>150}\,(CINEMA \infty SALLE))$$

$$\pi_{Cinema}\,(CINEMA \infty \sigma_{Capacité>150}\,(SALLE)\,)$$

Soit une jointure suivie d'une sélection, ou l'inverse.

NB: on peut les représenter comme des arbres.



L'optimisation sur un exemple

On suppose qu'il n'y a que 5% de salles de plus de 150 places.

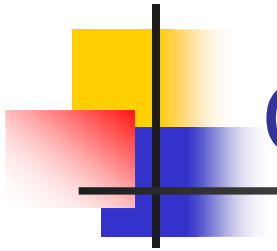
1. Jointure : on lit 3 600 pages (120x30); Sélection : on recherche dans ces 3600 pages les tuples verifiant la condition (capacité>150).

   Nombre d'E/S ≈ 3 600

2. Sélection : on lit 120 pages et on obtient 6 pages. Jointure : on lit 180 pages (6x30).

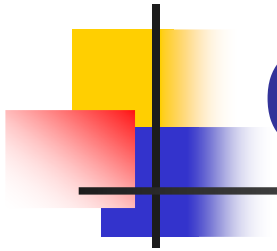
   Nombre d'E/S ≈ 120 + 180 = 300.

➔ la deuxième stratégie est de loin la meilleure !



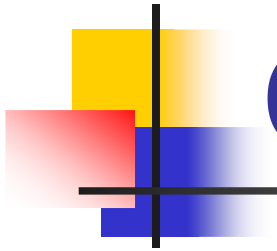
Conclusion sur l'optimisation logique

- La réécriture algébrique est nécessaire, mais pas suffisante
- Il faut tenir compte d'autres critères:
  – *Les chemins d'accès* aux données
- On peut accéder aux données d'une table par accès séquentiel, par index, par hachage, etc.
  – *Les différents algorithmes* possibles pour réaliser un opérateur
- Il existe par exemple plusieurs algorithmes pour la jointure
- Souvent ces algorithmes dépendent des chemins d'accès disponibles
  – *Les propriétés statistiques* de la base de données
- Taille des tables
- Sélectivité des attributs
- etc.



Optimisation physique

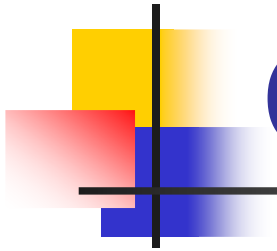
- Objectif: remplacer les opérateurs algébriques par des algorithmes (appelés également stratégies).
- Le choix de l'algorithme dépend entre autre de la taille des relations (nombre d'accès E/S) et de la prise en compte des chemins d'accès aux relations.



Optimisation physique

La performance d'une requête est évaluée en fonction:

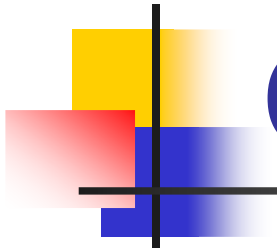
- *TempsES : Temps d'accès (lectures et écritures) à la mémoire secondaire*
- *TempsUCT : Temps de traitement de l'unité centrale (souvent négligeable* par rapport au temps d'accès)
- *TailleMC : Espace requis en mémoire centrale*
- *TailleMS : Espace requis en mémoire secondaire*



Optimisation physique

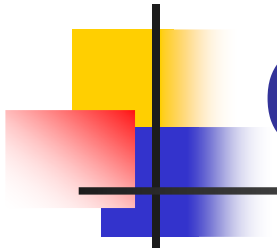
Dans les systèmes transactionnels:

- Le principal soucis est de pouvoir traiter les requêtes le plus rapidement possible
- Les accès disque sont les opérations les plus coûteuses
- Donc, la principale métrique de performance est *TempsES*



Optimisation physique

- le SGBD procède au calcul du coût d'une opération algébrique en fonction de la taille des relations, de l'algorithme choisi et des chemins d'accès.
- l'optimisation physique consiste à utiliser les connaissances statistiques stockées, et des connaissances sur les coûts d'implantation des opérateurs (algorithmes) afin de choisir le meilleur plan d'exécution physique.



Optimisation physique

- **Les données statistiques :**

- Pour optimiser la requête, l'optimiseur utilise les données statistiques de la BD mémorisées dans les catalogues telles que : la cardinalité des relations, le nombre de pages occupées par une relation, le nombre de niveaux d'index, le nombre de pages de chaque niveau, etc.

Optimisation physique

- L'estimation du coût des opérations se base sur des statistiques
- Statistiques d'une table *T:*
- N_T *Nombre de lignes de la table T*
- $TailleLigne_T$ *La taille d'une ligne de la table T*
- FB_T Facteur de blocage moyen de *T:* (nombre moyen de lignes contenues dans un bloc)
- FBM_T Facteur de blocage maximal de *T* , **Estimation : (*TailleBloc-TailleDescripteurBloc)/*** $TailleLigne_T$
- B_T *:* Nombre de blocs de la table *T,* **Estimation :** N_T / FB_T
- $Card_T(col)$*:* Nombre de valeurs distinctes (cardinalité) de la colonne *col*

pour la table *T,* Ex : $Card_T(sexe) = 2$

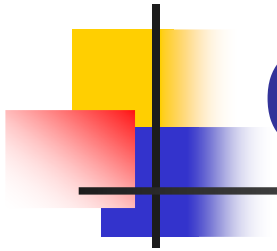
- $Min_T(col)$, $Max_T(col)$*:* Valeurs minimale et maximale que peut prendre une colonne *col*

Optimisation physique

- Statistiques d'évaluation

Statistiques d'une expression de sélection par valeur (*col = val) ou par* intervalle (*col* $\in [val_1, val_2]$) *:*

- *FacteurSélectivité*$_T$ *(col = val):* Pourcentage de lignes pour lesquelles la colonne *col* contient la valeur *val,* **Estimation : 1/** ***Card*$_T$ *(col)***
- *FacteurSélectivité*$_T$ *(col* $\in [val_1, val_2]$*):* Pourcentage de lignes pour lesquelles la colonne *col* contient une valeur comprise entre *val1 et val2,* **Estimation : (*val2- val1)/(MaxT (col)- MinT (col))***
- *SelT (col = val):* Nombre de lignes de *T sélectionnées par l'expression de* sélection, **Estimation :** ***FacteurSélectivité*$_T$ *(col) * N*$_T$**



Optimisation physique

- Opérations à optimiser

1) Sélection de lignes selon une clé
 – Ex: balayage, index, hashage, etc.

2) Tri des lignes d'une table
– Ex: algorithme tri-fusion

3) Jointure de deux tables
– Ex: boucles imbriquées, jointure tri-fusion, etc.

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de sélection :**
  - Sélection sans index :
- Le fichier est non trié sur l'attribut de restriction : le balayage séquentiel complet du fichier est alors nécessaire. Cout de l'opération=B_T, T : relation opérande, B_T : Nombre de blocs de T

- Le fichier est trié sur l'attribut de restriction : une recherche dichotomique est possible, cout de l'opération : $Log_2(B_T)$

Algorithmes d'implantation des opérateurs algébriques

Balayage de tables (FULL TABLE SCAN)

- Situations où il peut être acceptable de balayer une table:

– Petites tables: Plus avantageux de lire tous les blocs de la table en mémoire centrale et de faire les opérations (sélection, tri, jointure, etc.) en mémoire centrale

– Tables intermédiaires:

- Ex: table retournée par un SELECT imbriqué
- On ne possède pas d'index ou autres structures d'optimisation pour ces tables
- Le balayage est souvent la seule option

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de sélection :**
- Sélection avec index de type arbre B+ :Sélection par égalité dans un index secondaire (INDEX SCAN)

  un index associe les valeurs d'un attribut avec la liste des adresses de tuples ayant cette valeur. Souvent l'index est organisé en arbre B+, dans ce cas le cout de l'opération= nombre de niveaux de l'arbre (B+) +1 accès au bloc contenant la valeur recherchée (cas de la restriction attribut=valeur)

Algorithmes d'implantation des opérateurs algébriques

- Remarques:
- – L'index secondaire est préférable au balayage si la colonne de sélection possède plus de 200 valeurs différentes (*Cardinalité(col) ≥ 200)*
- – Oracle recommande de NE PAS créer d'index si la sélection peut retourner plus que 15% des lignes (*FacteurSélectivité(col) ≥ 15%), facteurSélectivité(col)=1/cardinalité(col)*

– Ex: créer un index sur une colonne *sexe est une mauvaise idée*

Algorithmes d'implantation des opérateurs algébriques

- Balayage de l'index (FULL INDEX SCAN)

Dans les cas où toutes les colonnes souhaitées se trouvent dans la clé de l'index

Ex: (index sur la colonne *idClient d'une table Transaction)*

```
SELECT idClient, COUNT(*) as nbTransactions
FROM Transaction
GROUP BY idClient
```

- On balaye les blocs feuilles de l'index au lieu de balayer les blocs de la table
- Comme l'information est plus compacte dans les feuilles de l'index, il y a moins de blocs à lire

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de jointure:**

- La jointure est l'opération la plus coûteuse , son optimisation est très importante

- Plusieurs algorithmes, dépendants du chemin d'accès Chacun peut être meilleur dans des situations spécifiques – Choix entre plusieurs algorithmes - meilleure optimisation

- Principaux algorithmes: Boucles imbriquées simples, Tri-fusion, Jointure par hachage, Boucles imbriquées avec index

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de jointure :** boucles imbriquées

```
Pour chaque bloc B1 de R1 faire
        Lire (R1, B1) ;
        Pour chaque bloc B2 de R2 faire
                Lire (R2, B2) ;
                Pour chaque tuple tuple1 de R1 faire
                        Pour chaque tuple tuple2 de R2 faire
                                Si tuple1 et tuple2 sont joignable
                        alors écrire (Resultat, tuple1+tuple2) ;
                                fin
                        fin
                fin
fin.
```

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de jointure :** boucles imbriquées

Cout(opération)=B_{R1}+(B_{R1}*B_{R2})

(on suppose qu'on a en mémoire 2 bloc: 1 bloc pour chaque table)

Si la mémoire permet de mémoriser (B+1) blocs, il est possible de garder B blocs de R1 et de lire R2 seulement B_{R1}/B fois, la formule devient alors:

Cout(opération)=B_{R1}+((B_{R1}/B)*B_{R2})

Si *R1* entre en mémoire alors le coût est B_{R1}+B_{R2}

– La jointure par boucles imbriquées est *inefficace* sur de grandes tables

– Toutefois, si l'une des relations entre en mémoire , elle est *très efficace*

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de jointure :** Algorithme de tri-fusion

- Principe: trier les deux relations suivant l'attribut de jointure, ensuite la fusion des tuples ayant même valeur est effectuée (parcours synchronisé des deux relations) (cas de l'equi jointure).

En supposant que le cout d'un tri (tri binaire) de N pages est $2N \text{Log}_2(N)$, le cout de la fusion $B_{R1} + B_{R2}$

Donc le cout de la jointure sera: $2* B_{R1} \text{Log}(B_{R1}) + 2* B_{R2} \text{Log}(B_{R2}) + B_{R1} + B_{R2}$

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de jointure :** jointure avec index
- (cas ou l'une des relations est indexée sur l'attribut de jointure par exemple R) dans ce cas il suffit de balayer la deuxième relation S et accéder au fichier index pour chaque tuple. En cas de succès, on procède à la concaténation pour composer le tuple résultat.
- Lorsque les deux relations sont indexée sur les attributs de jointure il suffit de fusionner les deux index, les couples des adresses des tuples concaténés sont alors obtenus.

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de jointure :** jointure avec index
- Le cout est de l'ordre de B_S+n*N_S, n étant le nombre d'accès en moyenne pour trouver un article dans le fichier indexé, et N_S est le nombre de lignes de S.

- Cet algorithme est efficace seulement si l'index est très sélectif (sélectivité de l'attribut proche de 1)

  Sélectivité de l'attribut R.B: $X_{R.B}$ = nombre de valeurs distinctes de R.B / |R|

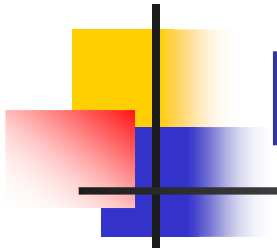
  1/ $X_{R.B}$ = le nombre moyen d'articles ayant une même valeur pour R.B

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de jointure :** jointure avec hachage
- Charger la relation R en mémoire
- Construire la table de hachage de R
- Parcourir la table S. Pour chaque ligne de la table S appliquer la fonction de hachage pour retrouver la partition de la table R contenant les lignes susceptibles de joindre avec elle.
- Pas de tri, pas besoin d'index

Algorithmes d'implantation des opérateurs algébriques

- **Opérateur de jointure :** jointure avec hachage
- Cout:
- Hachage: lecture *R* et pour chaque enregistrement il faut lire/écrire le bloc de hachage – Coût hachage = B_R lectures + |R| lectures + |R| écritures = B_{R1}+ 2|R|
- Jointure: lecture *S* et pour chaque article de *S* il faut lire un bloc de hachage – Coût jointure = B_S lectures + |*S*| lectures = B_S + |*S*|
- Coût total = B_R + B_S + 2|R| + |S|
- Il est préférable de faire le hachage sur la relation la plus petite
- Remarque: si *R* haché tient en mémoire , le coût se réduit à B_R + B_S



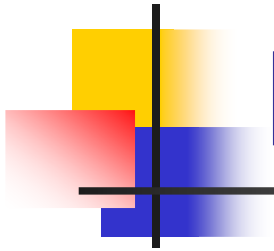
Exemple sous Oracle

FILM (Titre, Acteur, Realisateur)

VU (Titre, Spectateur)

Requête: acteurs et réalisateurs des films vus

```
SELECT ACTEUR, REALISATEUR
FROM FILM, VU
WHERE FILM.TITRE=VU.TITRE
```



Exemple sous Oracle

- Il n'existe pas d'index sur TITRE ni dans FILM ni dans VU,(algorithme de tri-fusion)

Plan d'exécution

---

0 SELECT STATEMENT

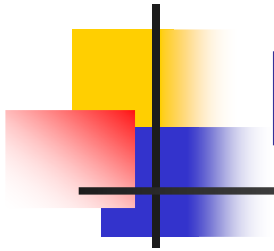
1 MERGE JOIN

2 SORT JOIN

3 TABLE ACCESS FULL VU

4 SORT JOIN

5 TABLE ACCESS FULL FILM



Exemple sous Oracle

Il existe un index sur TITRE dans FILM seulement. (balayage sequentiel de vu et accés à travers l'index pour film)

CREATE INDEX FILM_TITRE_idx on FILM (TITRE);

Plan d'exécution

---

0 SELECT STATEMENT

1 NESTED LOOPS JOIN

2 TABLE ACCESS FULL VU

3 TABLE ACCESS BY ROWID FILM

4 INDEX RANGE SCAN FILM_TITRE_IDX

Ex: Jointure par hachage

Considérez l'exemple suivant. Il extrait toutes les ventes pour les six derniers mois avec les détails correspondants de l'employé :

```
SELECT * FROM ventes s
JOIN employes e ON (s.id_supplementaire = e.id_supplementaire AND
s.id_employe = e.id_employe) WHERE s.date_vente > trunc(sysdate) - INTERVAL '6' MONTH
```

Le filtre DATE_VENTE est la seule clause where indépendante. Cela signifie que cela fait référence à une seule table et n'appartient pas aux prédicats de jointure.

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		49244	59M	12049
* 1	**HASH JOIN**		49244	59M	12049
2	TABLE ACCESS FULL	EMPLOYES	10000	9M	478
*** 3**	TABLE ACCESS FULL	VENTES	49244	10M	10521

Predicate Information (identified by operation id):

1-access("S"."ID_SUPPLEMENTAIRE"="E"."ID_SUPPLEMENTAIRE"AND "S"."ID_EMPLOYE" ="E"."ID_EMPLOYE")

3-filter("S"."DATE_VENTE">TRUNC(SYSDATE@!) -INTERVAL'+00-06' YEAR(2) TO MONTH)

Ex: Jointure par hachage

- Id : Identifiant de l'opérateur ;
- Operation : type d'opération utilisée
- Name : Nom de la relation utilisée ;
- Cost: coût estimé par oracle;
- Rows : Le nombre de lignes qu'Oracle pense transférer. ,
- Bytes : Nombre d'octets qu'oracle pense transférer.

Ex: Jointure par hachage

La première étape de l'exécution est un parcours complet de la table pour charger tous les employés dans une table de hachage (identifiant 2 du plan). La table de hachage utilise les prédicats de jointure comme clé. À l'étape suivante, la base de données fait un autre parcours complet de la table VENTES et ignore toutes les ventes qui ne satisfont pas la condition sur DATE_VENTE (identifiant 3 du plan). Pour les enregistrements VENTES correspondants, la base de données accède à la table de hachage pour charger les détails correspondants des employés.

Le seul but de la table de hachage est d'agir comme une structure temporaire en mémoire pour éviter d'avoir à accéder de nombreuses fois à la table EMPLOYE. La table de hachage est initialement chargée en une seule fois, pour qu'il ne soit pas nécessaire d'utiliser un index pour récupérer efficacement des enregistrements seuls. L'information du prédicat confirme qu'aucun filtre n'est appliqué sur la table EMPLOYES (identifiant 2 du plan). La requête n'a aucun prédicat indépendant sur cette table.

CONCLUSION

- L'intérêt de l'optimisation logique:

✓ Le coût des opérateurs varie en fonction du volume des données traitées, c'est-à-dire plus le nombre de tuples des relations traitées est petit, plus les coûts CPU et d'E /S sont minimisées.

✓ Certains opérateurs diminuent le volume de données telles que la restriction et la projection.

- L'intérêt de l'optimisation physique :

✓ minimiser les coûts lors de l'implémentation des opérateurs algébriques en fonction des caractéristiques des fichiers supportant les relations concernées par les opérateurs.