***Cheat sheet SGBD 3à5 :***

Problème :

EMP(ENO, ENAME, TITLE)

PROJECT(PNO, PNAME, BUDGET)

WORKS(ENO, PNO, RESP, DUR)

Soit la requête :

pour chaque projet de budget > 250 qui emploie au moins 2 employés, donner le nom et le titre des employés

Comment l’exprimer en SQL ?

• Avec count : traduire en algèbre ?

• Sans count : plus complexe (variables Evaluation et optimisation - 2 2 Sans count : plus complexe (variables supplémentaires)

Avec count :

SELECT DISTINCT Ename, Title

FROM Emp, Project, Works

WHERE Budget > 250 AND Emp.Eno=Works.Eno AND Project.Pno=Works.Pno AND Project.Pno IN (SELECT Pno FROM Works GROUP BY Pno HAVING COUNT (\*) >= 2);

Sans count :

SELECT DISTINCT E1.Ename, E1.Title

FROM Emp E1, Project, Works W1, Emp E2, Works W2

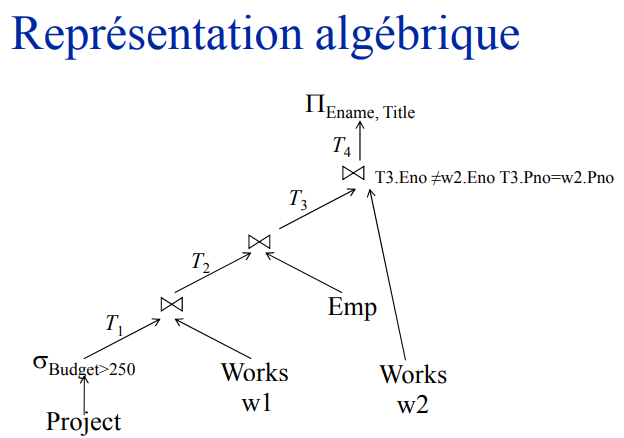
WHERE Budget > 250

AND E1.Eno=W1.Eno AND E2.Eno = W2.Eno

AND Project.Pno=W1.Pno AND W2.Pno = W1.Pno

AND E1 Eno <> E2 Eno; Evaluation et optimisation - 3 E1.Eno <> E2.Eno;

Plan d’exécution possible (sans count) :



T1 <- lire la table projet et sélectionner les tuples Budget<250

T2 <- Joindre T1 avec la relation Works

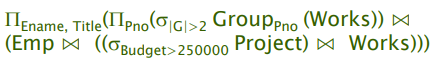
T3 <- Joindre T2 avec la relation Emp

T4 <- Joindre T3 avec Works sur Pno en vérifiant

T5 <- Projeter T4 sur Ename,Title

RESULTAT <- Eliminer les doublons dans T5

Représentation algébrique : les bonnes questions :



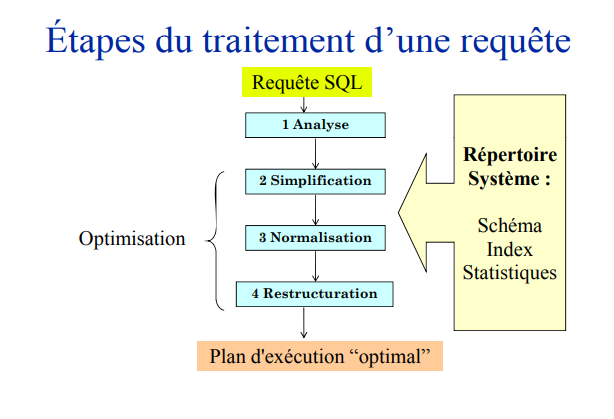
Comment obtenir ces plans ?

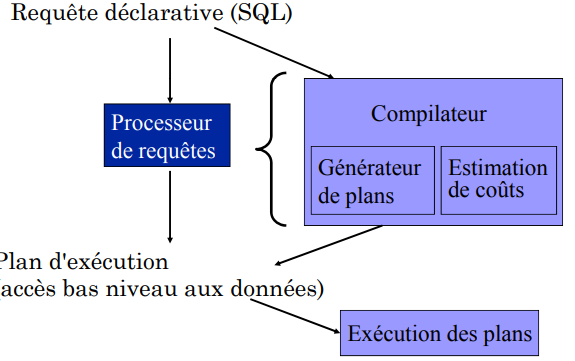
Comment exécuter chaque nœud/sous-arbre ?

Quel est le coût ?

Quel est le meilleur plan pour une requête donnée ?

Traitement des requêtes :





le

Normalisation des requêtes :

* Analyse lexicale et syntaxique :
  + Vérification de la validité de la requête
  + Vérification des attributs et relations
  + Vérification du typage de la qualification
* Mise de la requête en forme normale :
  + Forme normale conjonctive

(p11∨p12∨…∨p1n) ∧…∧ (pm1∨pm2∨…∨pmn)

* + Forme normale disjonctive (

(p11∧p12 ∧…∧p1n) ∨…∨ (pm1 ∧pm2∧…∧ pmn)

* + OR devient union
  + AND devient jointure ou sélection

Simplification :

Pourquoi simplifier ?

Plus une requête est simple plus son exécution peut être efficace

Comment ? en appliquant des transformations

* Elimination de la redondance
  + Règle d’idempotence
  + Application de la transitivité
* Eliminer des opérations redondantes :
  + ex. : pas besoin de distinct après une projection sur une clé
* Utilisation des règles d’intégrité

**Alternative de traduction :**

Select Ename

From Emp e , Work w

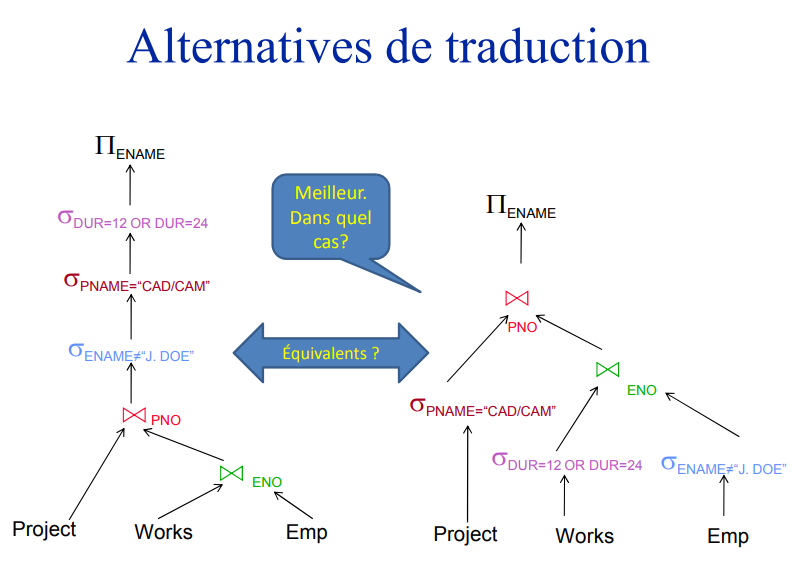
Where e.Eno = w.eno

And w.Dur > 37

* Stratégie 1 :
* Stratégie 2 :

La stratégie 2 semble « meilleur » car elle évite un produit cartésien et sélectionne un sous ensemble de Works avant la jointure

Comment mesurer la qualité d’une stratégie ?



**Estimer le coût d’un plan :**

La fonction de coût donne une estimation des temps I/O et CPU :

Nombre d’instructions et accès disques (écriture/lecture en nb pages)

1. Estimation du nombre d’accès disque pendant l’évaluation de chaque nœud de l’arbre algébrique
2. Estimation de la taille du résultat de chaque nœud par rapport à ses entrées :

Sélectivité des opérations – « facteur de réduction » influe sur la taille du résultat = opérande du prochain opérateur basé sur les statistiques maintenues par le SGBD

Deux hypothèses fortes :

1. Hypothèse d’uniformité : les différentes valeurs d’un attribut ont la même probabilité.
2. Hypothèse d’indépendance des attributs : la probabilité d’un attribut ne dépend pas de la proba d’un autre

**Tailles des relations intermédiaires :**

Projection :

Produit cartésien :

Union :

Borne sup :

Borne inf :

Différence :

Borne sup : /\*

Borne inf  : /\*RS

Jointure :

.cas particulier : A est clé de R et B est la clé étrangère dans S vers R

.plus généralement

Sélection :  
 taille(R)=card(R) x largeur(R)

Ou est une estimation de la sélectivité du prédicat, dont la forme générale est « taille des sélectionnés/tailles des possibles (domaine) »

|  |  |
| --- | --- |
|  | |
|  |  |
|  | |
|  | |
|  | |

**Type d’organisation :**

Organisation séquentielle :

* Non trié :
  + Très facile à maintenir en mise à jour
  + Parcourir toutes les pages quelque soit la requête
* Trié :
  + Un peu plus difficile a maintenir
  + Parcours raccourci car on peut s’arrêter dès qu’on a les données cherchées

Organisations indexées :

* Objectifs
  + Accès rapide a partir d’un clé de recherche
  + Accès séquentiel trié ou non
* Moyens
  + Utilisation d’indexe permettant la recherche de l’adresse de l’enregistrement à partir d’une clé de recherche
* Exemple
  + Dans une bibliothèque, recherche des ouvrages par thème, par auteur, ou par titre.
  + Dans un livre, rechercher les paragraphes contenants tel mot ?

**Les index :**

Entrée d’un index :

* On appelle une entrée la structure qui associe une clé de recherche avec l’adresse des enregistrements concernés
* Trois alternatives pour la structure d’une entrée
  1. Entrée d’index contient les données
  2. Entrée d’index contient (k,ptr)
  3. Entrée d’index contient (k, liste de ptr)

Index non plaçant :

* Les index non plaçant sont dit secondaires
* Index=structure auxiliaire en plus des données quelle que soit la façon dont elle sont stockées
  + Données stockées sans être triées
  + Données triées selon un attribut autre que celui indexé
* Définir un Index non plaçant en SQL
  + create index ‘NOM’ on ‘TABLE(ATTRIBUTS)’ ;
  + Exemple create index IndexAge on Personne(Age)

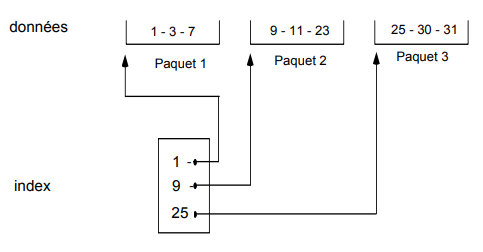
Index plaçant :

* Le stockage des données est organisé par l’index
* Les enregistrements ayant la même valeur de clé sont juxtaposé
  + Stockages contigus dans un paquet et dans les paquets contigus suivant si nécessaire
* Définir l’organisation des données lors de la création de la table
  + create table…organisation index : données triées selon la clé primaire
  + create cluster … : données regroupées par valeur d’un attribut
* Un entrée contient les données (cf :alternative1)
* Evidemment, pas plus d’un index plaçant par table

Index plaçant non dense :

Un index contenant toutes les clés est dit dense

* Un Index non dense concerne seulement les index plaçants
  + Les données doivent être stockées triées
* Objectif : obtenir un index occupant moins de place



* Méthode : enlever des entrées. Ne garder que les entrées nécessaires pour atteindre le bloc de données contenant les enregistrements recherchés
  + Garder l’entre ayant la plus petite/grande clé de chaque page
  + Ne pas indexe deux fois la meme clé sur 2 pages consécutives
* Inconvénient : toutes les valeurs de l’attribut indexé ne sont pas dans l’index.

Accès aux données par un index :

Sert pour évaluer une sélection

Index couvrant une requête :

* Un index (non plaçant) couvre une requête s'il est possible d‘ « évaluer la requête » sans lire les données
* Tous les attributs mentionnés dans la requête doivent être indexés
* Index couvrant une sélection
  + Pour chaque prédicat p de la clause where, il faut un index capable d'évaluer p.
* Index couvrant une projection
  + Pour chaque attribut de la clause select, il faut un index dense
* Avantage
  + Evite de lire les données, évaluation plus rapide d'une requête
* Concerne seulement les index non plaçant :
  + Un index plaçant contenant les données, elles sont forcément lues.

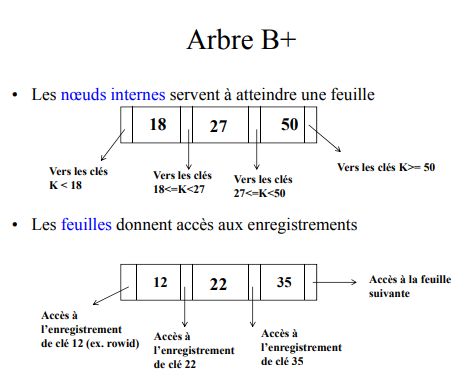
**Méthode d’accès :**

Arbre B+ :

* Les arbres B+ sont des index hiérarchiques
* Ils améliorent l’efficacité des recherches
  + L’arbre est peu profond
    - Accès rapide a un enregistrement
    - Rmq : l’arbre peut être très large, sans inconvénient
  + L’arbre est toujours équilibré
    - Balanced tree en anglais
    - Tous les chemins de la racine aux feuilles ont la même longueur
  + L’arbre est suffisamment compact
    - Peut souvent tenir en mémoire
    - Un nœud est au moins à moitié rempli

Arbre B+ : coût d’accès :

* Le cout d’accès



* + est proportionnel à la longueur d’un chemin
  + Identique quelle que soit la feuille atteinte

->Le coût d’accès est prévisible

* Avantage :
  + Permet d'estimer le coût d'accès, a priori, pour décider d'utiliser ou non un index
* Mesure du coût :
  + Nombre de nœud lus/écrits
  + Nombre de pages de données lues / écrites

Arbre B+ :3 types de nœuds (cas non plaçant) :

* Racine
  + Point d’entrée pour une recherche
* Nœud intermédiaire
  + Peut contenir une valeur pour laquelle il n’existe aucun enregistrement
* Feuille :
  + Les Feuilles contiennent toutes les clés pour lesquelles il existe un enregistrement
  + Les feuilles contiennent uniquement des clés de la BD

Ordre d’un arbre, degré d’un nœud :

* Soit n le nombre de clés contenues dans un nœud
* Un arbre B+ est d’ordre d si
  + Pour un nœud intermediaire et une feuille :
  + Pour la racine :
  + Tout nœud doit dans une page :
    - 2d correspond à une page complétement remplie
* Degré sortant d’un nœud
  + Un noued intermédiaire (et la racine) ayant n valeur de clé a n+1 pointeur vers ses fils
  + Une feuille n’a pas de fils
* à part la racine, toute page est entre moitié pleine et complètement pleine

Hauteur min et max d’un arbre B+ :

* Hauteur minimum quand l’arbre est plein : h = log2d+1(N) +1
  + 2d clés par nœud
  + N feuilles (il y’a donc N\*2d clés s dans la table correspondante)
* Hauteur maximum quand les nœuds sont le plus vides possible : h = logd+1(N) +2
  + d clés par nœud, 1 pour la racine
  + 2N feuilles (2N\*d clés dans la table correspondante)
* En pratique d est assez grand, donc h reste raisonnable (4 pour une relation « normale »