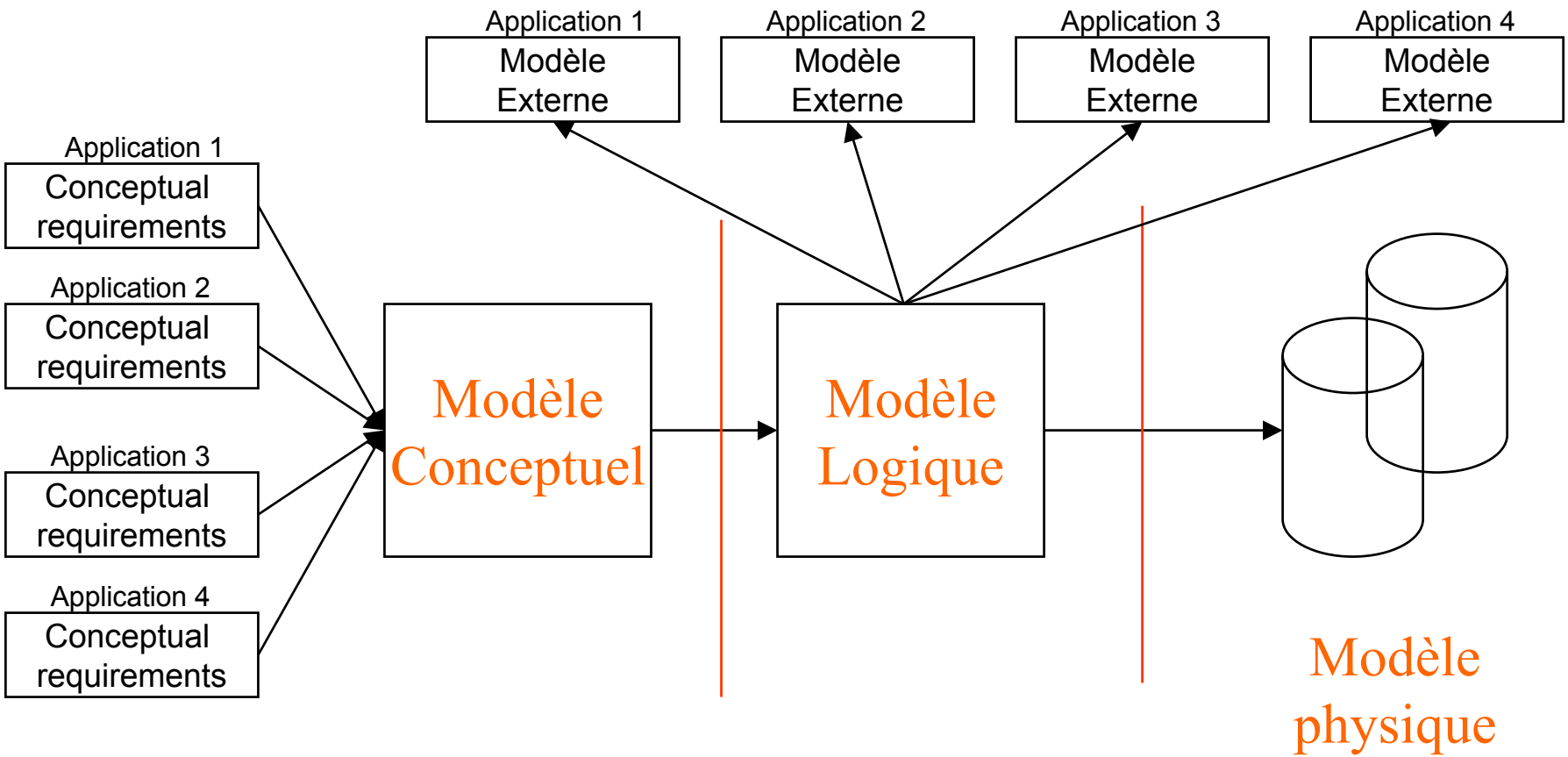
Structures d'Accès & Optimisation de Requêtes

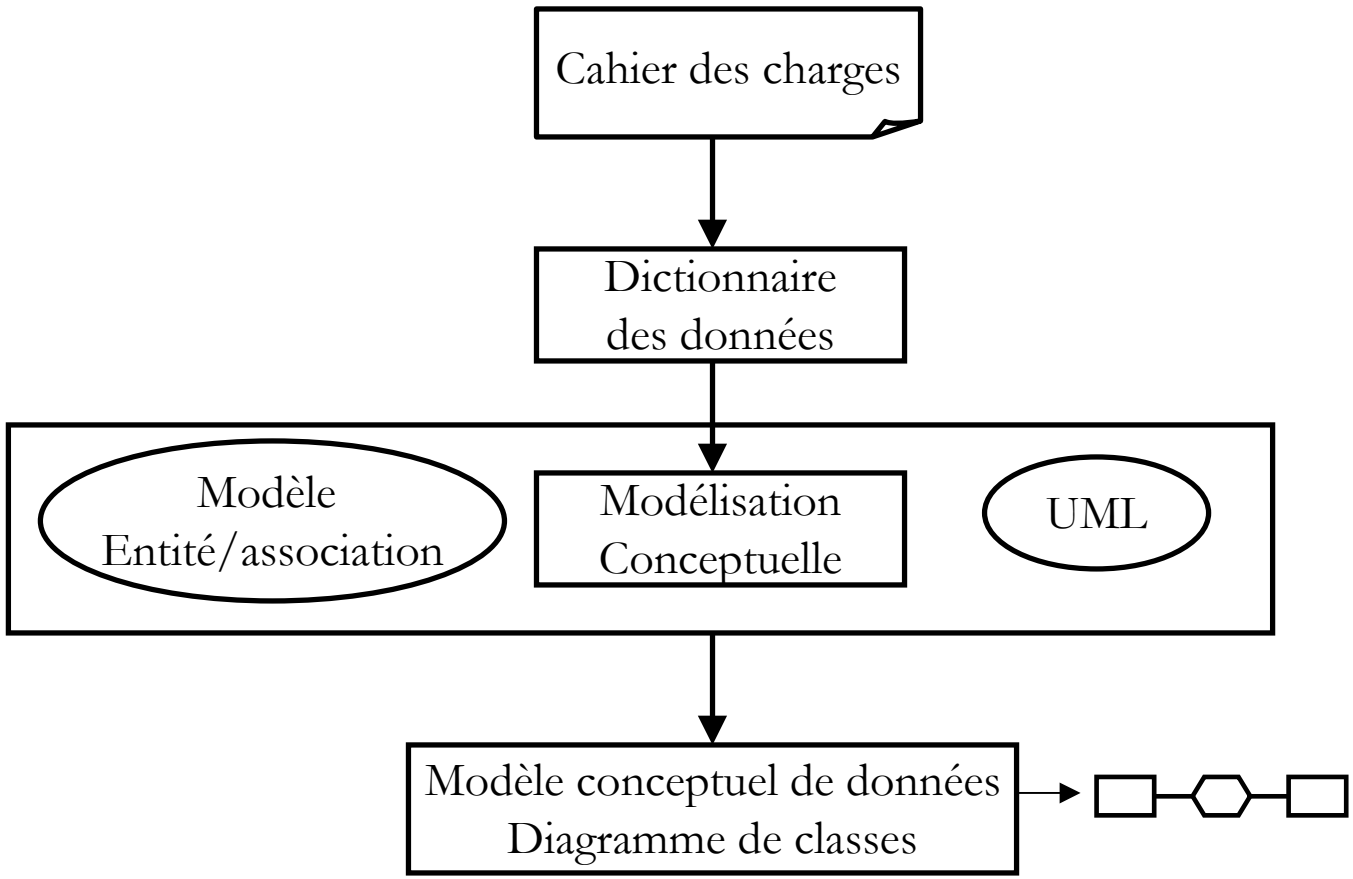
Ladjel BELLATRECHE

bellatreche@ensma.fr
05 49 49 80 72

Processus de conception d'une base de données



Modèle conceptuel



Modèle logique

- Passage du modèle conceptuel de données vers le modèle relationnel ou relationnel objet
  - Les deux règles: père - fils, père - père
- Normalisation
  - 1FN, 2FN, 3FN, etc.
    - ➡Tables normalisées

Conception physique

- Le but : le **traitement efficace** des données (une BD est censée contenir des milliers voire des millions d'enregistrements)
- Optimiser les performances des services de base de données: demand speedy access to data, **no matter how complex the queries are**
- La conception physique nécessite des informations manipulées hors de la modélisation conceptuelle et logique

Informations nécessaires pour la conception physique

- Relations normalisées et leurs tailles estimées
- Définitions de chaque attribut
- Types d'opérations sur les données et leurs fréquences
  - interrogation, suppression, modification, insertion
- Contraintes: temps de réponse, stockage, sécurité, reprise après les pannes, contraintes d'intégrité, etc.
- Descriptions de la technologie utilisée pour l'implémentation de la base de données (Oracle, SQL Server, etc.)

Plan

- Classification des requêtes
- Méthodes d'accès
  - Techniques d'indexation
- Optimisation des requêtes
  - Expressions algébriques
  - Opérations algébriques

Typologie des requêtes (I)

- Exemple d'une BD:

  Ouvrier (NSS, Nom, Salaire, Spec, #N°U)

  Usine (N°U, Site, Prod, #NSSChef)

  Contrat(N°Cont, Fin, Cout, #NSS, #N°U)

- Classification des requêtes: [Shasha et al. ``Database tuning: A principled approach", Prentice Hall, 1992]

  1. Requête singulière (Point query)
     - Requête basée sur une seule relation et dont le prédicat de sélection est composé avec une ou plusieurs égalités de manière à ce que la réponse comprenne au plus un tuple.
  - Exemple:

    SELECT * FROM Ouvrier WHERE NSS = 1233333;

Typologie des requêtes (II)

- Requête singulière multiple (multipoint query)
  - Requête basée sur une seule relation dont le prédicat de sélection est composé d'une ou plusieurs égalités, et dont la réponse comprend plusieurs tuples
  - Exemple:

```
SELECT * FROM Ouvrier WHERE Salaire > 1500 AND Spec =`soudeur ';
```

- Requête d'intervalle (range query)
  - Requête basée sur une seule relation et dont le prédicat comprend un test d'intervalle.
  - Exemple:

```
SELECT * FROM Ouvrier WHERE Salaire > 1500 AND Salaire < 5000;
```

Typologie des requêtes (III)

- Requête d'appartenance à un ensemble ordonné d'attributs Z (prefix match query)
  - Requête dont les conditions d'égalités doivent inclure obligatoirement une sous-chaîne définie selon l'ensemble Z
  - Exemple: Z = {NSS, Nom}

    SELECT Nom, Salaire FROM Ouvrier WHERE NSS = 12345 AND Nom = `Dupont';
- Requête min-max (Extermal query)
  - Requête formulée sur une seule relation et dont le prédicat est composé d'égalités faisant référence à une valeur minimale ou maximale.
  - Exemple: SELECT Nom, Salaire FROM Ouvrier WHERE Salaire = max (SELECT salaire FROM Ouvrier);

Typologie des requêtes (IV)

- Requête de groupement (Grouping query)
  - Requête dont la réponse est partitionnée par un ou plusieurs attributs
  - Exemple:

    SELECT N°U, AVG(Salaire) FROM Ouvrier GROUP BY N°U;

- Requête de jointure (join query)
  - Requête basée sur plusieurs relations.
  - Exemple:

    SELECT NSS, Nom, Site FROM Ouvrier O, Usine U WHERE O.N°U = U.N°U;

Techniques d'indexation

- Arbre B (B-tree)
- Bitmap

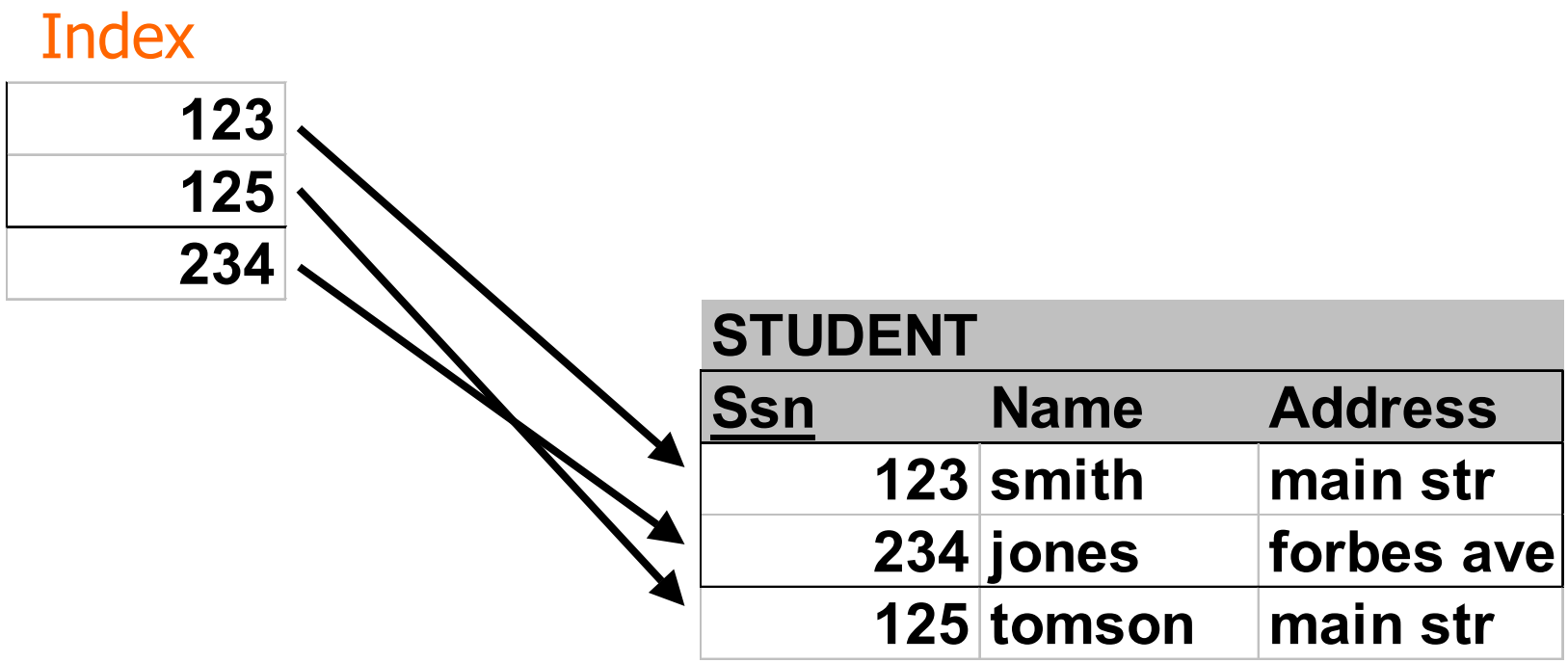
Index

- Une fois les tuples stockés dans un fichier (disque), comment faites vous la recherche d'une manière efficace? (eg., ssn=125?)

STUDENT		
Ssn	**Name**	**Address**
123	smith	main str
234	jones	forbes ave
125	tomson	main str

1. Accès séquentiel
2. Accès direct : utilisation d'index

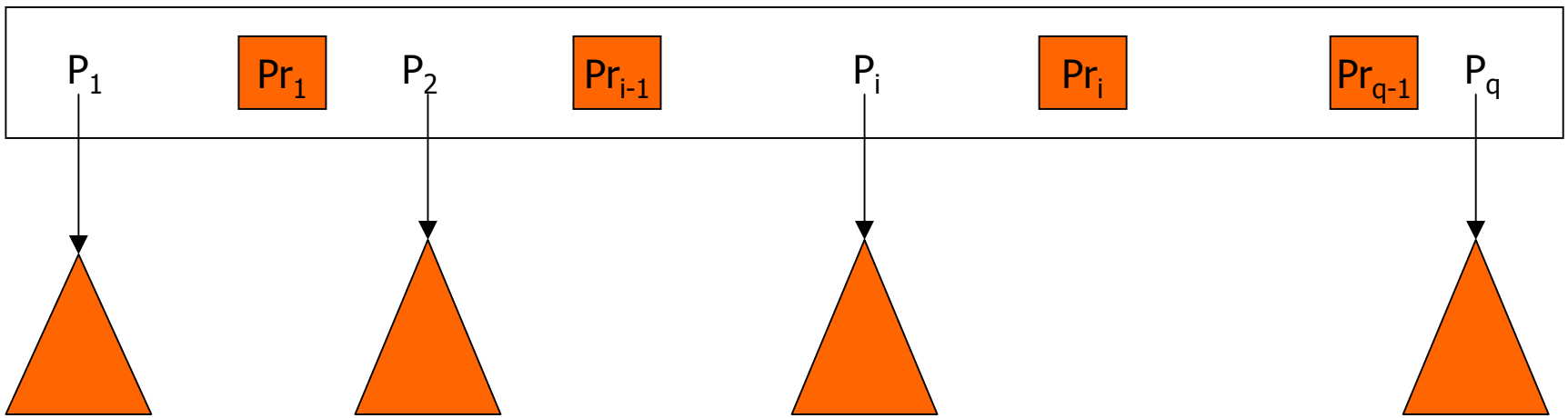
Index – idée principale



Arbre B

- Un arbre équilibré
- Chaque nœud est un index local
- Il se réorganise dynamiquement
- Utilisé par tous les SGBDs commerciaux

Nœud d'un arbre B

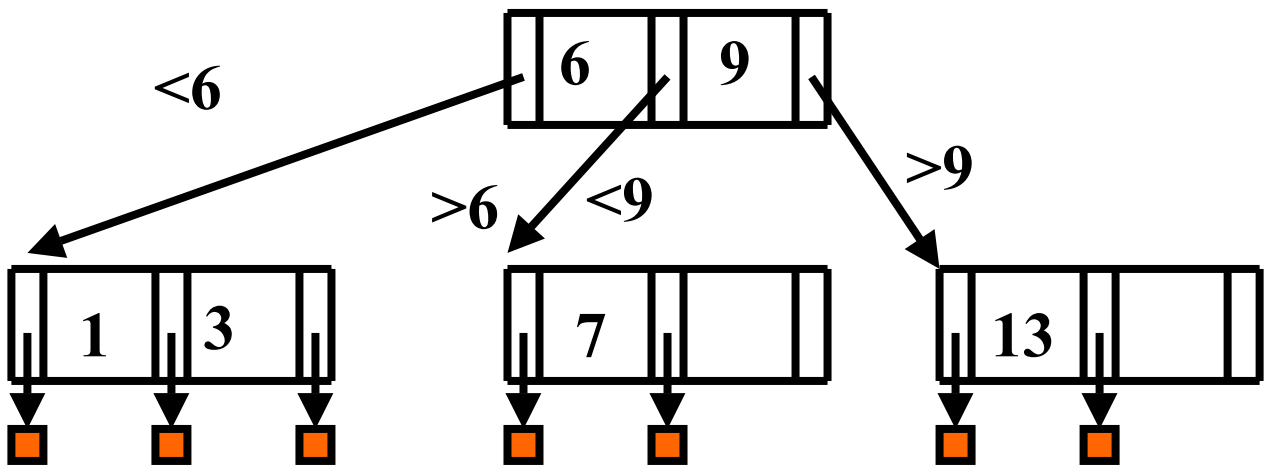


Sous-arbre pointé par P1

<P1, <Pr1>, P2, <Pr2>, …, <Prq-1>, Pq>

- q ≤p
- Chaque Pi est un pointeur vers un sous-arbre
- Pri est un pointeur sur les données
- Chaque nœud a au plus p pointeurs d'arbre

Exemple

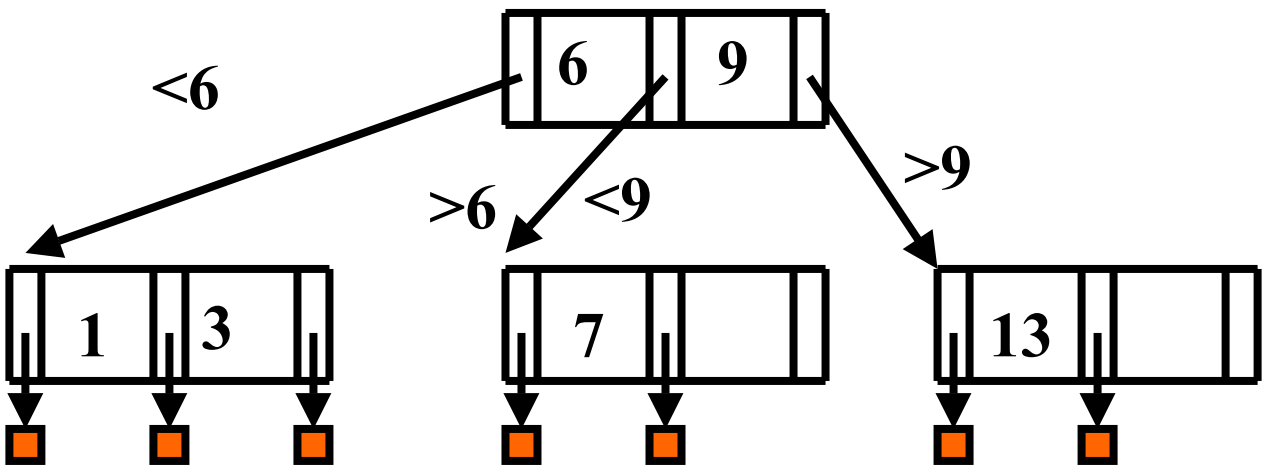


Properties

- “block aware” nodes: each node -> disk page
- O(log (N)) for everything! (ins/del/search)
- Utilization >= 50%, guaranteed; on average 69%

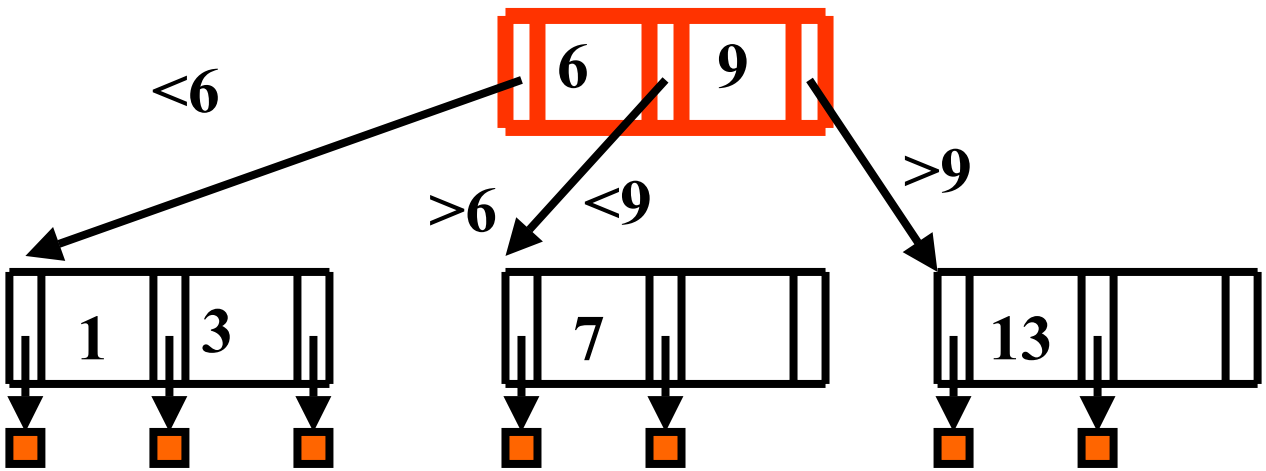
Requêtes

- Scénario pour les requêtes singulières
  - Prédicat de sélection: SSN=8



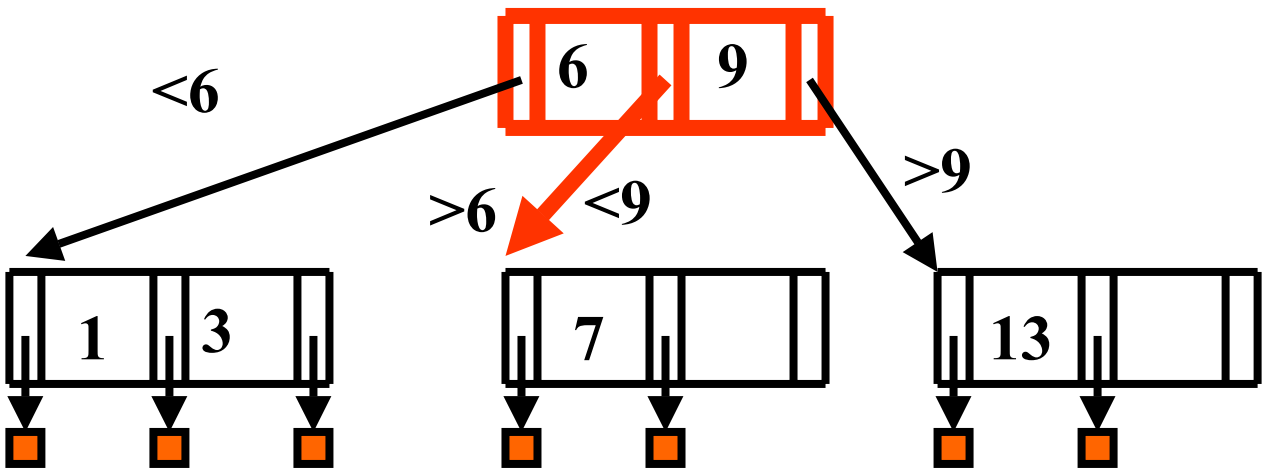
Requêtes

- Scénario pour les requêtes singulières
  - Prédicat de sélection: SSN=8



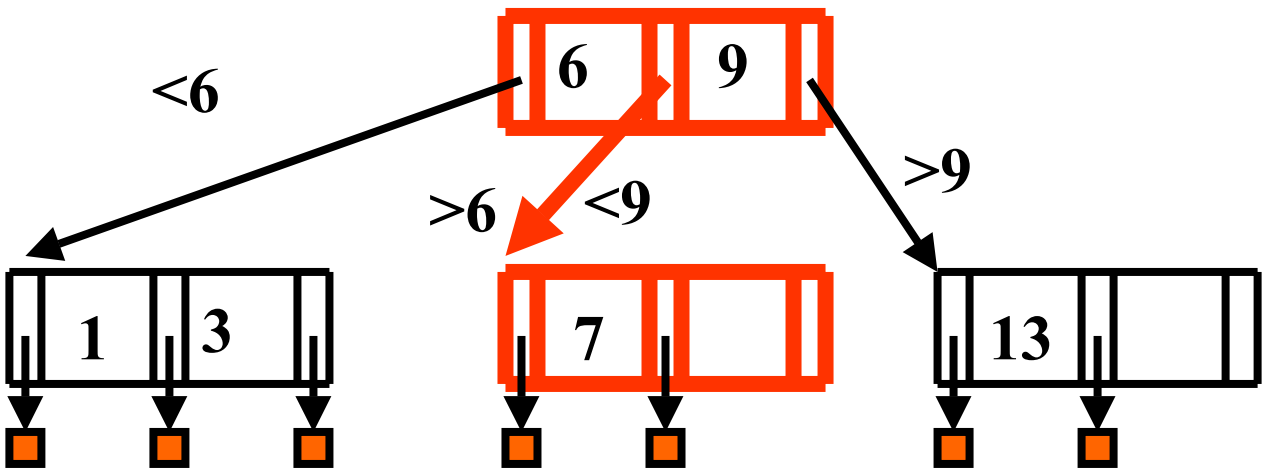
Requêtes

- Scénario pour les requêtes singulières
  - Prédicat de sélection: SSN=8



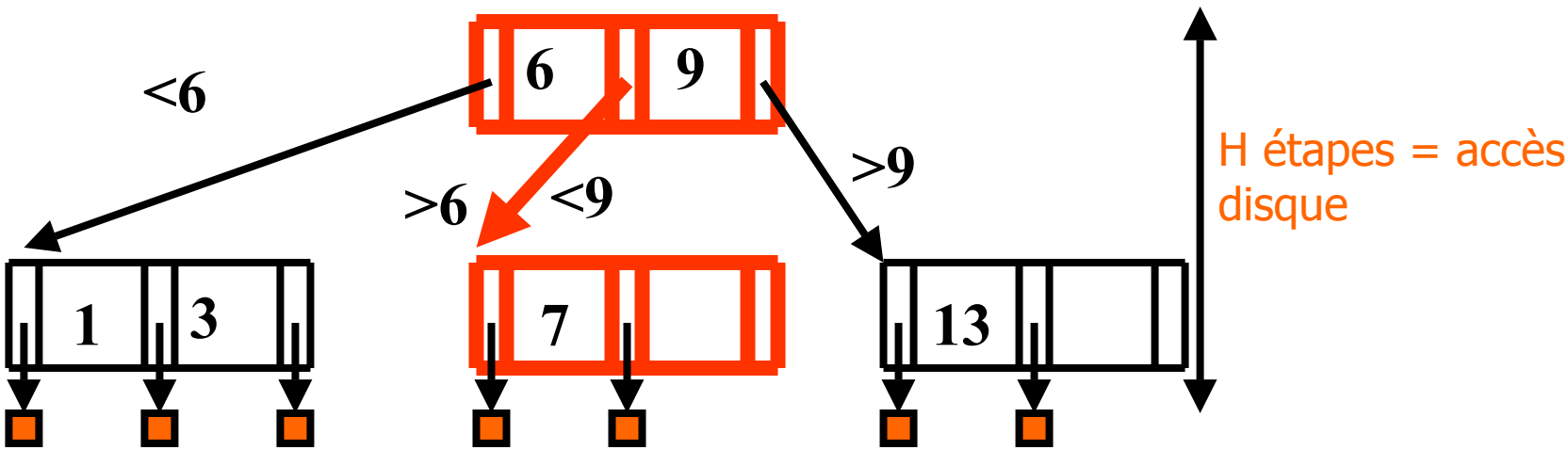
Requêtes

- Scénario pour les requêtes singulières
  - Prédicat de sélection: SSN=8



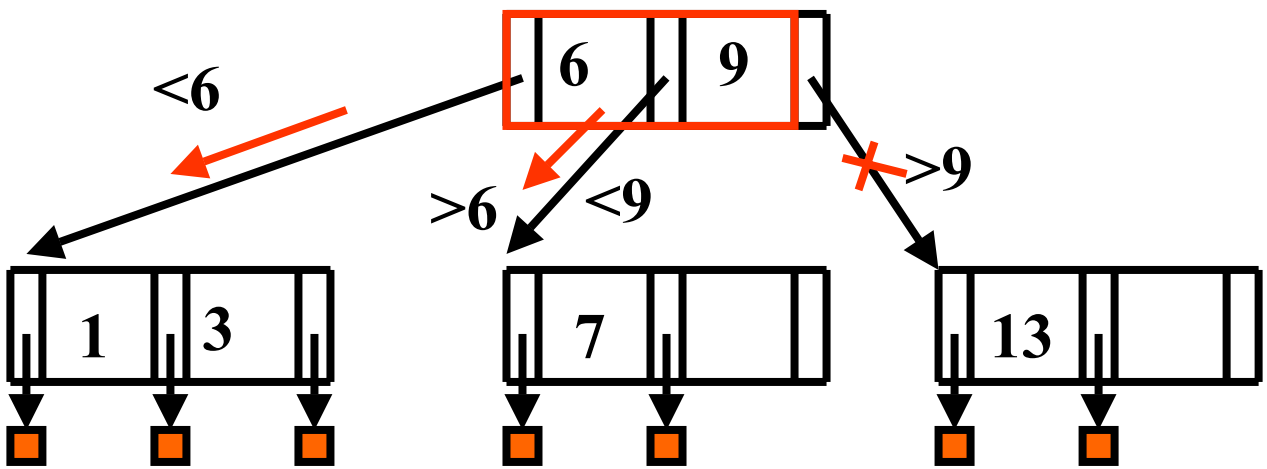
Requêtes

- Algorithme d'accès



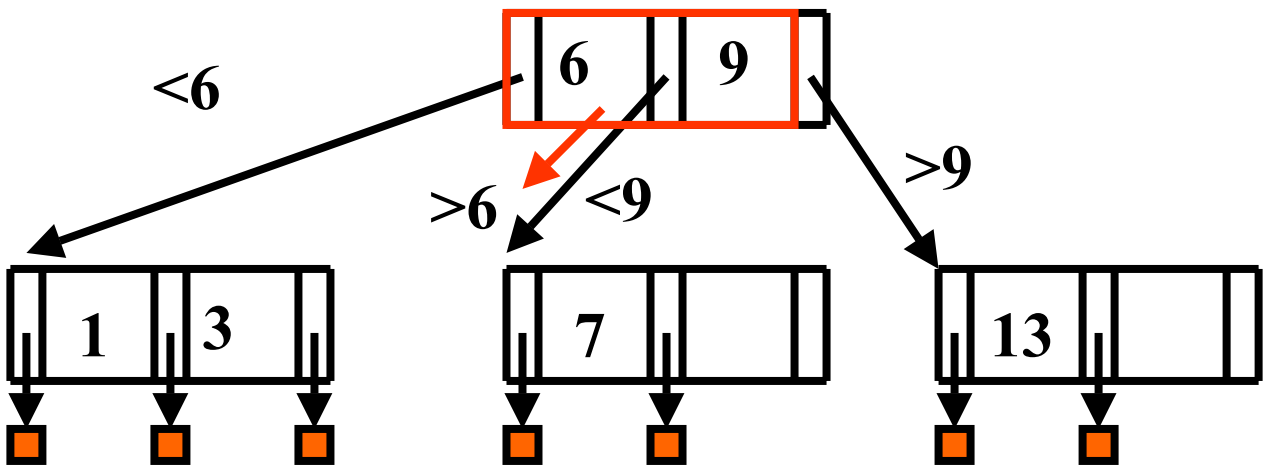
Requêtes d'intervalle

- Exemple: 5<salaire<8
- Proximity/ nearest neighbor searches? (eg., *salaire ~ 8*)



Requêtes d'intervalle

- Exemple: 5<salaire<8
- Proximity/ nearest neighbor searches? (eg., *salaire ~ 8*)



Recherche dans une table avec un index

- Exemple: SELECT * FROM Etudiant WHERE Age = 24;
- Un index sur Age permet d'obtenir rapidement les RIDs des tuples des employés de 24 ans.
- Requêtes conjonctives

  SELECT * FROM Etudiant WHERE Salaire = 2000 AND Ville = `Poitiers ';

  - Supposons l'existence de deux index; un sur Salaire et l'autre sur Ville.
  - Le premier index fournit les RIDs satisfaisant la 1ère condition qui seront regroupés dans une table temporaire T1. Idem pour le deuxième index ~ T2
  - Enfin l'intersection de T1 et T2 donne la réponse

Requêtes conjonctives ~ Index

- Attention:
  - Cette solution pourrait être inefficace si les facteurs de sélectivité des prédicats sont inférieurs à 0.6
- Dans un tel cas, le SGBD pourrait préférer faire un balayage séquentiel de la table

Index bitmap

- Problème:
  - Comment indexer une table sur un attribut qui ne prend qu'un petit nombre de valeurs?

Principe de l'index bitmap

- Soit un attribut A, prenant n valeurs possibles [v_1, …,v_n] (domaine)
- Création d'un index bitmap sur l'attribut A:
  - On crée n tableaux de bit, un pour chaque valeur v_i
  - Le tableau contient un bit pour chaque tuple t
  - Le bit d'un tuple t est à 1 si: t.A = v_i, à 0 sinon

Exemple

ROWID(RID)	Soudeur	Fraiseur	Sableur	Tourneur
00055 :000 :0023	0	1	0	1
00234 :020 :8922	1	0	0	0
19000 :328 :6200	0	0	0	1
21088 :120 :1002	0	0	1	0

Ouvrier(NSS, Nom, Salaire, Spécialité, N°U)
avec le domaine de l'attribut Spécialité est: {Soudeur, Fraiseur, Sableur, Tourneur}

SQL: CREATE INDEX BITMAP ON Ouvrier(Specialite) TABLESPACE spec_espace

Intérêt de Bitmap: Recherche

- Soit la requête suivante:

  SELECT * FROM Ouvrier WHERE Spec = ‘’Soudeur”

  - On prend le tableau pour la valeur Soudeur
  - On garde toutes les cellules à 1
  - On accède aux enregistrement par l’adresse

- ➡ Très efficace si le nombre de valeurs (n) est petit
  - Exemple: Sexe

Autre exemple

SQL: CREATE INDEX BITMAP ON Ouvrier(Salaire) TABLESPACE Sal_espace

ROWID(RID)	20000	30000	40000
00055 :000 :0023	1	0	0
00234 :020 :8922	0	1	0
19000 :328 :6200	0	0	0
21088 :120 :1002	0	0	1

Requête: Quels sont les employés soudeurs qui gagnent 30 000

Réponse: RID: 00234:020:8922

Quelle est la procédure?

Exemple avec count()

- Requêtes type OLAP (data warehouse)

  SELECT COUNT(*) FROM Ouvrier WHERE Spec in ('Soudeur', 'Tourneur')

- On compte le nombre 1 dans la colonne Soudeur
- On compte le nombre 1 dans la colonne Tourneur
- On fait la somme et c'est terminé

Les choix d'Oracle

- Par défaut l'index est un arbre B
  - Dès qu'on utilise une commande PRIMARY KEY, Oracle crée un arbre B sur la clé primaire
  - Arbre est stocké dans un segment d'index
- Index bitmap
- Hachage

Exemples de création d'index

- Le nom d'un index est choisi par le DBA et peut être normalisé en le préfixant par idx_ suivi du nom de l'attribut indexé et terminé par le nom de la table
- Exemple:

```
CREATE UNIQUE INDEX idx_NSS_Ouvrier ON Ouvrier(NSS);
CREATE UNIQUE INDEX idx_NSS_Salaire_Ouvrier ON Ouvrier(NSS, Salaire);
CREATE INDEX BITMAP ON Ouvrier(Specialite) TABLESPACE spec_espace
```

- Suppression d'un index

```
DROP INDEX <Nom de l 'index>;
DROP INDEX idx_NSS_Ouvrier;
```

Guide d'utilisation des index en cours d'exploitation d'une BD

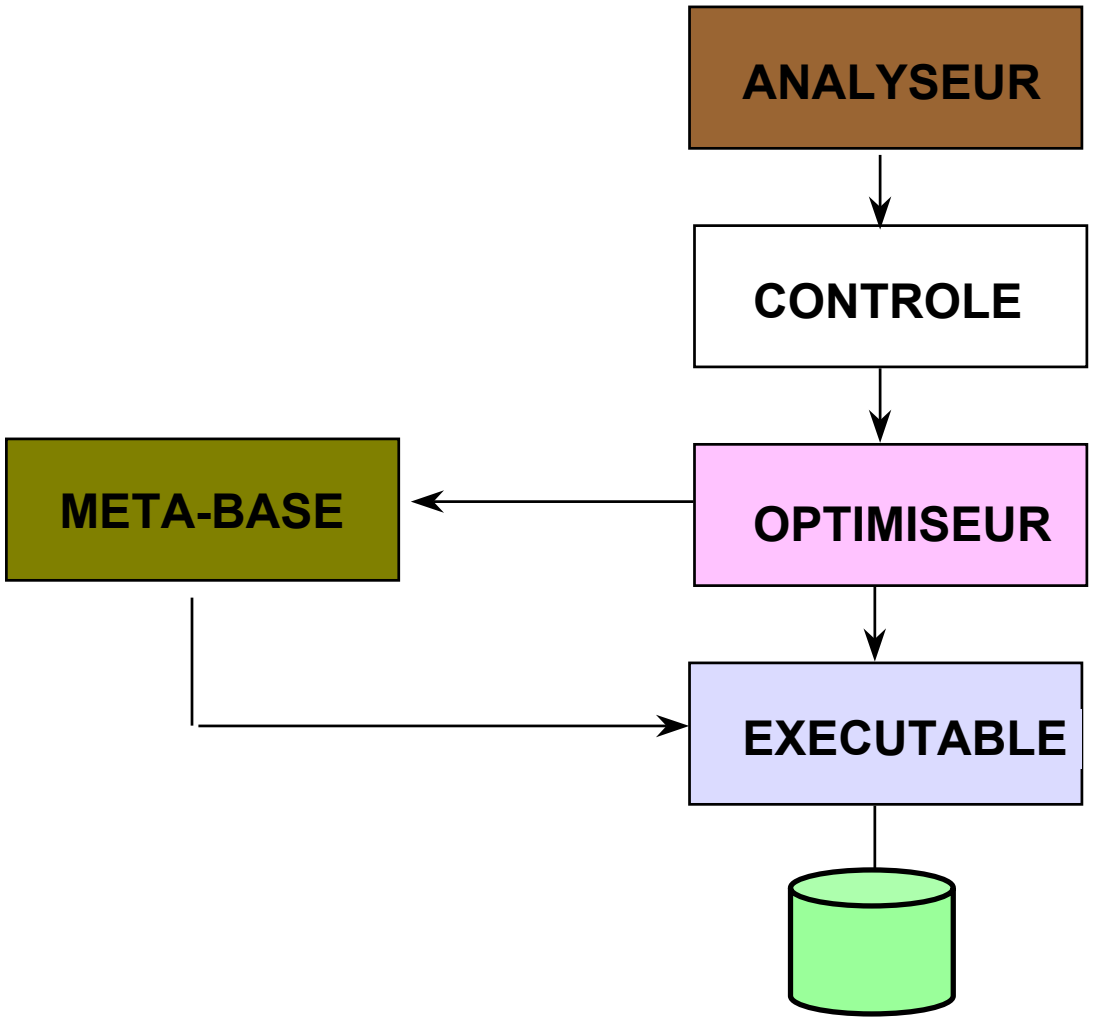
1. Les index utiles sont ceux nécessaires à l'exploitation courante des données
2. Supprimer les index lorsqu'il s'agit de traiter un flux important de transactions de type mise à jour sur une table. Ils sont d'abord supprimés, puis recréés au besoin après les mises à jour
3. Une accélération de la jointure est possible par la création des index: indexer les attributs de jointure permettra un accès plus rapide

Guide d'utilisation des index en cours d'exploitation d'une BD (suite)

4. Tous les attributs d'un index composé doivent être présents dans la clause WHERE, peu importe leur ordre dans la clause, pour que l'optimiseur en tire profit
5. Si le premier attribut de l'index composé est présent dans le WHERE, l'index composé est utilisé. Sinon, il devient inutile
6. La négation dans une condition bloque l'usage des index composés: les valeurs hors domaine ne sont pas indexées
7. Quand faut-créer un index composé?
   - Lorsque certains attributs sont fréquemment utilisés ensemble, il peut être avantageux de définir quelques index composés

Optimisation de requêtes

Introduction



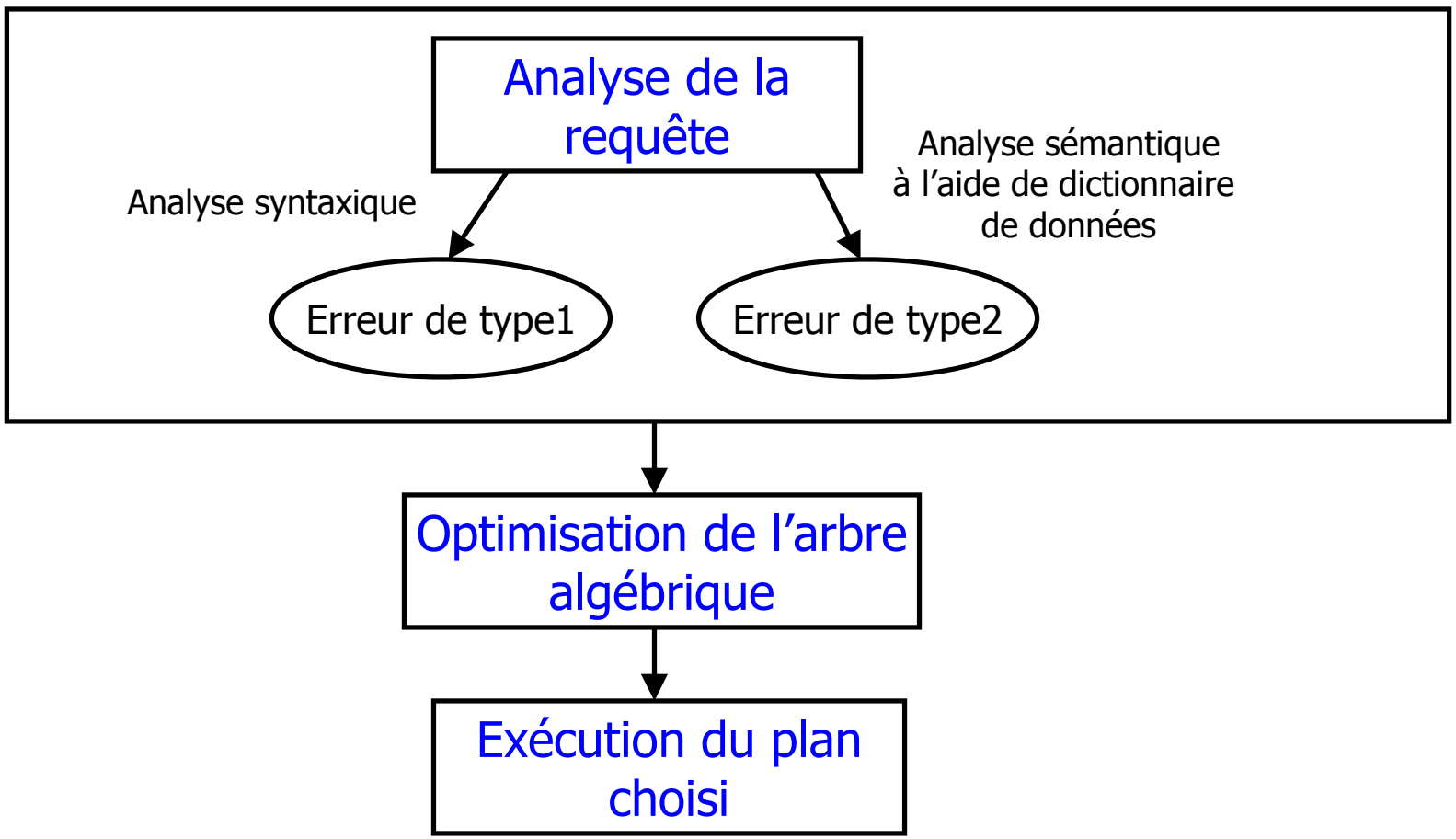
SYNTAXE
SEMANTIQUE
SCHEMA

VUES
INTEGRITE
AUTORISATIONS

ORDONNANCEMENT
ELABORATION
D'UN PLAN

EXECUTION
METHODES D'ACCES

Traitement d'une requête SQL



Arbre algébrique

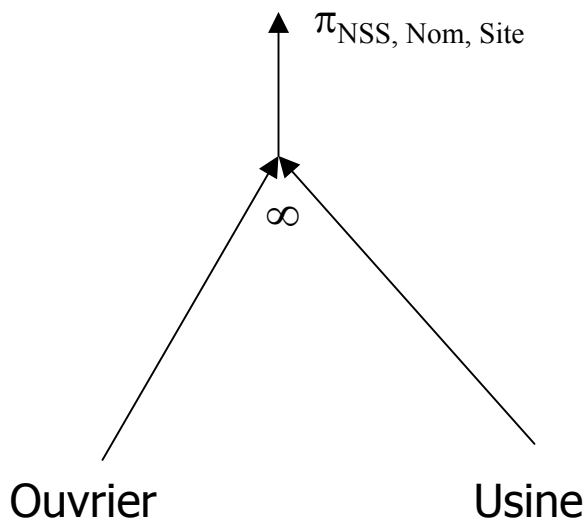
- **algèbre relationnelle** : modèle formel permettant d'exprimer et de calculer les requêtes sur les relations.
  - Constituée d'un ensemble d'opérateurs algébriques. Les plus utilisés sont la sélection, la jointure, la projection et les opérateurs ensemblistes.
  - Le résultat de l'exécution d'un opérateur est toujours une relation, ce qui permet la composition.
  - Utilisée également pour l'optimisation de requêtes.
  - Une requête algébrique peut se présenter sous forme d'un arbre algébrique.
  - Arbre algébrique : visualisation d'une requête sous une forme graphique d'arbre plus facile à lire qu'une forme linéaire.

Arbre algébrique

Un arbre algébrique est un arbre représentant une requête avec:

- **Les nœuds feuilles sont les relations de base**
- **Les nœuds intermédiaires sont les opérateurs**
- **Le nœud racine est le résultat**
- **L'arc est un flux de données**

Exemple: SELECT NSS, Nom, Site FROM Ouvrier O, Usine U WHERE O.N°U = U.N°U;

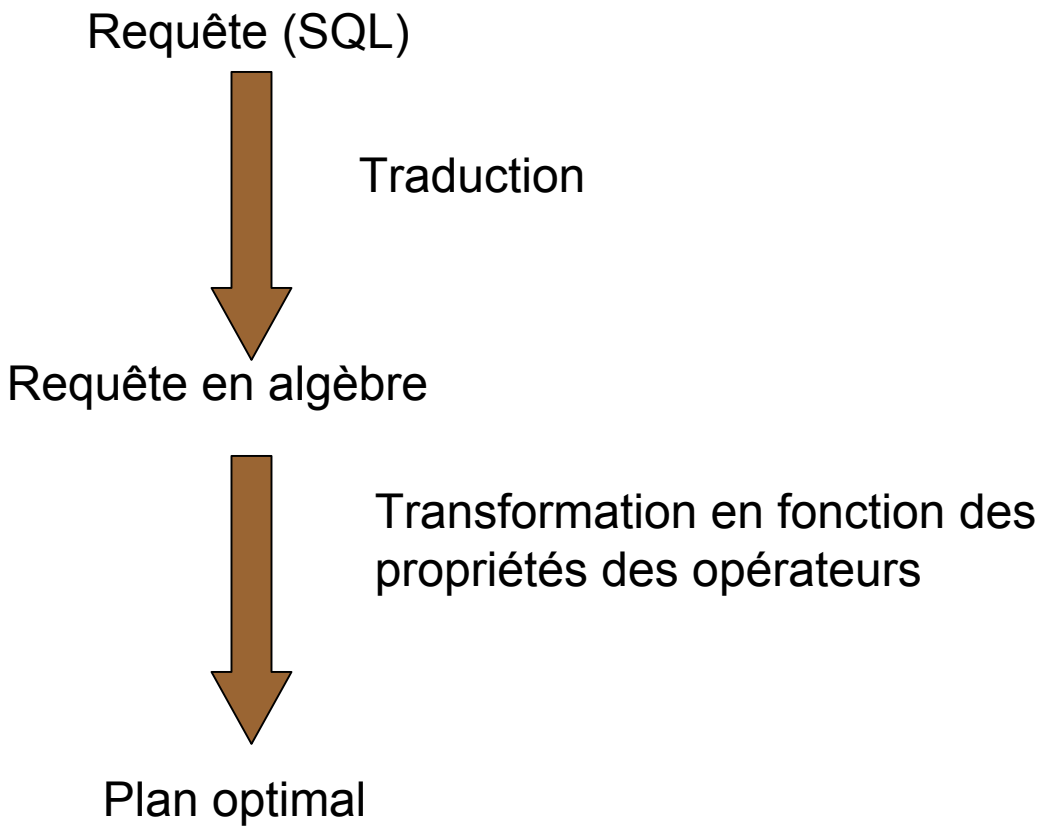


Techniques d'optimisation

1. Expressions algébriques:
   - Transformation des requêtes pour obtenir la meilleure séquence d'opérations
     - Approche qui marche bien (mais pas toujours)
2. Opérations algébriques:
   - Coût des différentes stratégiques possibles en fonction des caractéristiques des fichiers sur lesquels sont implantées les relations

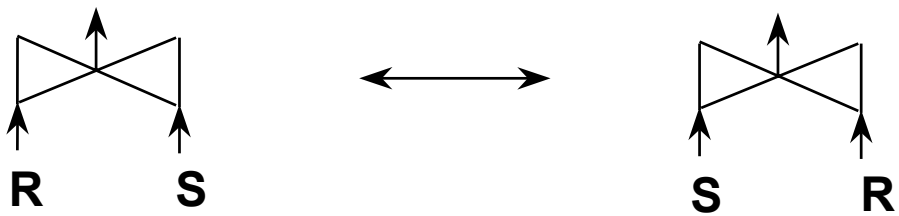
- Généralement les SGBDs commerciaux combinent les deux stratégies

Technique basée sur expressions algébriques

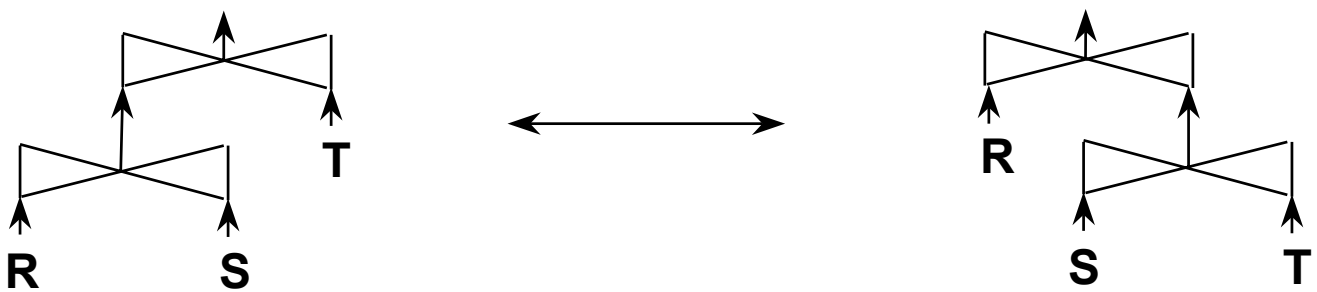


Propriétés des opérateurs

- Commutativité pour jointure et produit



- Associativité pour jointure et produit



➡ Il existe N!/2 arbres de jointure de N relations.

➡ Parmi les jointures, certaines sont des produits cartésiens.

Propriétés des opérations (suite)

- Groupage des projections

  $\pi_{A1, \ldots, An}(\pi_{B1, \ldots, Bm}(E)) = \pi_{A1, \ldots, An}(E)$ --- $\{A_1, \ldots, A_n\}$ inclus dans $\{B_1, \ldots, B_m\}$

- Exemple
  - $\pi_{nom, prénom}(\pi_{[age}$ (Étudiant))?
  - $\pi_{nom, prénom}(\pi_{nom, prénom, age}$ (Étudiant))
- Groupages de sélections

  $\sigma_{F1}(\sigma_{F2}(E)) = \sigma_{F1 \wedge F2}(E)$

- Exemple
  - $\sigma_{age>20}(\sigma_{sexe='F'}$ (Étudiant)) = $\sigma_{(age>20) \wedge (sexe='F')}$(Étudiant)
- Inversion σ et π
  - $\pi_{A1, \ldots, An}(\sigma_F(E)) = \sigma_F(\pi_{A1, \ldots, An}(E))$ si F porte sur $A_1, \ldots, A_n$
  - $\pi_{A1, \ldots, An}(\sigma_F(E)) = \pi_{A1, \ldots, An}(\sigma_F(\pi_{A1, \ldots, An, B1, \ldots, Bn})(E))$
    si F porte sur $B_1, \ldots, B_m$ qui ne sont pas parmi $A_1, \ldots, A_n$

Propriétés des opérations (suite)

- Inversion la sélection et le produit

  $\sigma_F (E_1 * E_2) = \sigma_F (E_1) * E$ si F ne porte que sur les attributs de E_1

  $\sigma_F (E_1 * E_2) = \sigma_{F1} (E_1) * \sigma_{F2} (E_2)$

  Si $F = F_1 \wedge F_2$ et F_i ne porte que sur les attributs de E_i

  $\sigma_F (E_1 * E_2) = \sigma_{F2} (\sigma_{F1} (E_1) * E_2)$ si F_1 porte sur les attributs de E_1 et F_2 sur ceux de E_1 et E_2

- Exemple
  - $\sigma_{sexe = 'F'}$(Étudiant * Cours) = $\sigma_{sexe = 'F'}$(Étudiant) * Cours
- Inversion sélection et union
  - $\sigma_F (E_1 \cup E_2) = \sigma_F(E_1) \cup \sigma_F(E_2)$
- Inversion sélection et différence
  - $\sigma_F (E_1 - E_2) = \sigma_F(E_1) - \sigma_F(E_2)$
- Inversion projection et produit
- Inversion projection et union

Principes d'optimisation des expressions algébriques

1. Exécuter les sélections aussitôt que possible
2. Réduire la taille des relations à manipuler
   - ➡ Combiner les sélections avec un produit cartésien pour aboutir à une jointure
3. Combiner des séquences d'opérations unaires (σ, π)
   - ➡ Cascade d'opérations appliquées groupée à chaque tuple
4. Mémoriser les sous-expression commune

Exécuter les sélections aussitôt que possible

- Appliquer ces règles (remplacer membre gauche par droit)

1. $\sigma_F (E_1 \cup E_2) = \sigma_F(E_1) \cup \sigma_F(E_2)$
2. $\sigma_F (E_1 - E_2) = \sigma_F(E_1) - \sigma_F(E_2)$
3. $\sigma_F (E_1 * E_2) = \sigma_F (E_1) * E_2$ si F ne porte que sur les attributs de E_1
4. $\sigma_F (E_1 * E_2) = \sigma_{F1} (E_1) * \sigma_{F2} (E_2)$ si $F = F_1 \wedge F_2$ et F_i ne porte que sur les attributs de E_i
5. $\sigma_F (E_1 * E_2) = \sigma_{F2} (\sigma_{F1} (E_1) * E_2)$ si F_1 porte sur les attributs de E_1 et F_2 sur ceux de E_1 et E_2

Combiner les sélection avec un produit cartésien pour aboutir à une jointure

- Transformation si possible du produit cartésien en jointure:
  - Si R * S est l'argument d 'une sélection σ_F alors si F est une comparaison entre attributs de R et de S, transformer R * S en jointure : $\sigma_F (R * S) = (R \infty_F) S$
- Attention : si F est une expression ne portant que sur les attributs de R:

$$\sigma_F (R * S) = \sigma_F (R) * S$$

Combiner des séquences d'opérations unaires

- Un ensemble d'opérations unaires peuvent être combinées et appliquées ensemble sur chaque tuple manipulé

Sous-expressions communes à une expression

- Si une expression contient plusieurs fois la même sous-expression
- Si cette sous-expression peut être lue en moins de temps qu'il faut pour la calculer
  - ➡ Matérialiser la sous-expression commune
- Principe de vues matérialisées (voir le cours sur les data warehouse)

Un algorithme d'optimisation

1. Séparer chaque sélection $\sigma_{F1 \wedge \ldots \wedge Fn}(E)$ en une cascade $\sigma_{F1}(\sigma_{F1 \ \ldots}(\sigma_{Fn}(E)))$
2. Descendre chaque sélection le plus bas possible dans l'arbre algébrique
3. Descendre chaque projection aussi bas possible dans l'arbre algébrique
4. Combiner des cascades de sélection et de projection dans une sélection seule, une projection seule, ou une sélection suivie par une projection
5. Regrouper les nœuds intérieurs de l'arbre autour des opérateurs binaires (*, -, $\cup$). Chaque opérateur binaire crée un groupe
6. Produire un programme comportant une étape pour chaque groupe, à évaluer dans n'importe quel ordre mais tant qu'aucun groupe ne soit évalué avant ses groupes descendants.

Exercice

- Soient trois relations: Livre, Prêt et Lecteur

  Livre(ISBN, titre, auteur, éditeur)

  Prêt(N°carteP, ISBNP, date)

  Lecteur(N°carte, nom, adresse)

- Requête: Liste des noms des lecteurs et des titres des livres pour tous les prêts effectués avant le 15/6/1990
- Trouver le meilleur plan d'exécution de cette requête

Quelques problèmes

- La restructuration d'un arbre sous-entend souvent la permutation des opérateurs binaires
- Cette optimisation ignore la taille de chaque table, les facteurs de sélectivité, etc.
- Aucune estimation de résultats intermédiaires
- Exemple:

  (Ouvrier ∞ Usine) ∞ Contrat

  - Si cardinalité(Usine) <<<Cardinalité(Contrat) << Cardinalité(Ouvrier) alors l'expression suivante a des chances d'être plus performante: (Ouvrier ∞ Contrat) ∞ Ouvrier

➡ Optimisation basée sur les modèles de coût

Méthodes d'optimisation basées sur les modèles de coûts

«Cost-based Methods»

Optimisation des opérations: Notions (I)

- Les tables relationnelles sont stockées physiquement dans des fichiers sur disque
- Lorsqu 'on veut accéder aux données, on doit transférer le contenu pertinent des fichiers dans la mémoire
  - Fichier = séquence de tuples
  - Bloc (page) = unité de transfert de données entre disque et mémoire
  - Facteur de blocage = nombre de tuples d'une relation qui «tiennent» dans un bloc
  - Coût de lecture (ou écriture) d'une relation = nombre de blocs à lire du disque vers la mémoire (ou à écrire de la mémoire vers le disque)

Estimation du coût des opérations physiques

- TempsES : temps d'accès à mémoire secondaire (MS)
- TempsUCT
  - Souvent négligeable
- TailleMC : espace mémoire centrale
- TailleMS : espace mémoire secondaire

Modèle du coût d'une entrée-sortie en mémoire secondaire

Paramètre	Signification
TempsESDisque(*n*)	Temps de total transfert (lecture ou écriture) de *n* octets du disque
TempsTrans(*n*)	Temps de transfert des *n* octets sans repositionnement
TempsPosDébut	Temps de positionnement au premier octet à transférer (ex : 10 *ms*)
TempsRotation	Délai de rotation (ex : 4 *ms*)
TempsDépBras	Temps de déplacement du bras (ex : 6 *ms*)
TauxTransVrac	Taux de transfert en vrac (ex : 2MB/sec)
TempsESBloc	Temps de transfert d'un bloc (ex : 11 *ms*)
TempsTrans	Temps de transfert d'un bloc sans repositionnement (ex : 1 *ms*)
TailleBloc	Taille d'un bloc (ex : 2K octets)

Statistiques au sujet des tables

Statistique	Signification
$\|\|T\|\|$	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
$NDIST$	Nombre de valeurs distinctes de la colonne pour la table T
$Min_T(colonne)$	Valeur minimum de la colonne de T
$Max_T(colonne)$	Valeur maximum de la colonne de T
$\|T\|$	Nombre de pages de la table T

Facteur de sélectivité

- ||(σ(R))|| = SF * ||R|| avec:

  SF (A = valeur) = 1 / NDIST(A)

  SF(A > valeur) = (max(A) - valeur) / (max(A) - min(A))

  SF(A < valeur) = (valeur - min(A)) / (max(A) - min(A))

  SF(A IN liste valeurs) = (1/NDIST(A)) * CARD(liste valeurs)

  SF(P et Q) = SF(P) * SF(Q)

  SF(P ou Q) = SF(P) + SF(Q) - SF(P) * SF(Q)

  SF(not P) = 1 - SF(P)

- Le coût dépend de l'algorithme (index, hachage ou balayage).

Facteur de sélectivité

- La taille d'une jointure est estimée par la formule suivante:

$$||R1 \infty R2|| = Sel * ||R1|| * ||R2||;$$

avec sel: la sélectivité de la jointure

Exemple

```
SELECT *
FROM R1, R2
```

➡ **SF =1**

```
SELECT *
FROM R1
WHERE A = valeur
```

➡ **SF = 1/NDIST(A) avec un modèle uniforme**

Plans d’exécutions

Ce qu'il faut savoir

- Qu'est ce qu'un plan d'exécution
  - Un programme combinant des opérateurs physiques (chemins d 'accès et traitements de données)
  - Il a la forme d'un arbre: chaque nœud est un opérateur qui
    - prend des données en entrée
    - applique un traitement
    - produit les données traitées en sortie

Algorithmes Généraux pour les opérateurs relationnels

La sélection

- Sélection sans index

  SELECT NSS, Nom FROM Ouvrier WHERE Salaire > 15000

- Plan d'exécution

  pour chaque tuple t de ouvrier faire -- accès séquentiel

  si t.salaire > 15000 alors réponse = réponse $\cup$

  $\pi_{NSS,\ Nom}$ (Ouvrier)

Algorithmes de jointure

- Jointure sans index
  - Jointure par boucles imbriquées (nested loop)
  - Jointure par tri-fusion (sort-join)
  - Jointure par hachage (hash-join)
- Jointure avec index
  - Jointure avec boucles indexées

Jointure par boucles imbriquées (JBI)

- $R \infty_F S$
  - Principe:
    - La première table est lue séquentiellement et de préférence stockée entièrement en mémoire
    - Pour chaque tuple de R, il y a une comparaison avec les tuples de S

Jointure par boucles imbriquées

POUR chaque ligne l_R de R
    POUR chaque ligne l_S de S
        SI θ sur l_R et l_S est satisfait
            Produire la ligne concaténée à partir de l_R et l_S
        FINSI
    FINPOUR
FINPOUR

+ Simple
Complexité: (||R|| + (||R||*||S||)
Si la table extérieure tient en mémoire: une seule lecture des deux tables suffit.

Jointure par tri fusion

- Plus efficace que les boucles imbriquées pour les grosses tables
- Principe:
  - Trier le deux tables sur les colonnes de jointure
  - Effectuer la fusion
- Complexité: Le tri qui coûte cher
- Quelle est la complexité de cet algorithme?

Jointure par tri fusion

Trier R et S par tri externe et réécrire dans des fichiers temporaires
Lire groupe de lignes $G_R(c_R)$ de R pour la première valeur c_R de clé de jointure
Lire groupe de lignes $G_S(c_S)$ de S pour la première valeur c_S de clé de jointure
TANT QUE il reste des lignes de R et S à traiter
    SI $c_R = c_S$
        Produire les lignes concaténées pour chacune des combinaisons de lignes de $G_R(c_R)$ et $G_S(c_S)$;
        Lire les groupes suivants $G_R(c_R)$ de R et $G_S(c_S)$ de S;
    SINON
        SI $c_R < c_S$
            Lire le groupe suivant $G_R(c_R)$ de R
        SINON
            SI $c_R > c_S$
                Lire le groupe $G_S(c_S)$ suivant dans S
            FINSI
        FINSI
    FINSI
FIN TANT QUE

Jointure par hachage

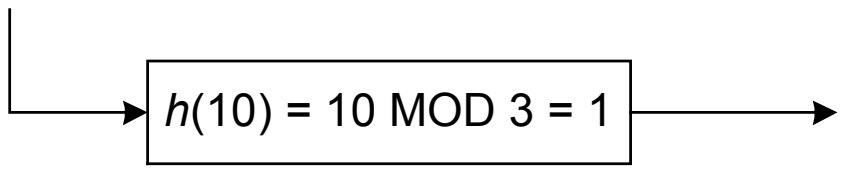
- Algorithme récent, encore peu répandu
- Très efficace quand une des deux tables est petite (1, 2, 3 fois la taille de la mémoire)
- Algorithme en option dans Oracle

Principe

- Hachage ou adressage dispersé (hashing)
- Fonction *h(clé de hachage)* ➡ l'adresse d'un **paquet**
- Fichier = tableau de *paquets* (***bucket***)
  - ~ ARRAY paquet [0..*TH*-1]
  - *TH* : *taille de l'espace d'adressage primaire*
- Habituellement paquet = bloc(page)
- Pas d'index à traverser : ***O***(1) en meilleur cas
- Sélection par égalité

Exemple

clé = 10



0	60	Erable argenté	15.99
	90	Pommier	25.99
	81	Catalpa	25.99
1	70	Herbe à puce	10.99
	40	Epinette bleue	25.99
	10	Cèdre en boule	10.99
2	20	Sapin	12.99
	50	Chêne	22.99
	95	Génévrier	15.99
	80	Poirier	26.99

Hachage vs indexage

- **O**(1) en meilleur cas vs **O**(log(N))
- Pas d'espace supplémentaire d'index
- Gaspillage d'espace si *TH* trop grand
- Performance dégradée si TH trop petit
- Gestion plus délicate
  - La détermination de la fonction de hachage h et TH
  - Maintenance : réorganisations

Principe de la jointure par hachage

1. On hache la plus petite des deux tables en n fragments
2. On hache la seconde table, avec la même fonction, en n autres fragments
3. On réunit les fragments par paire, et on fait la jointure

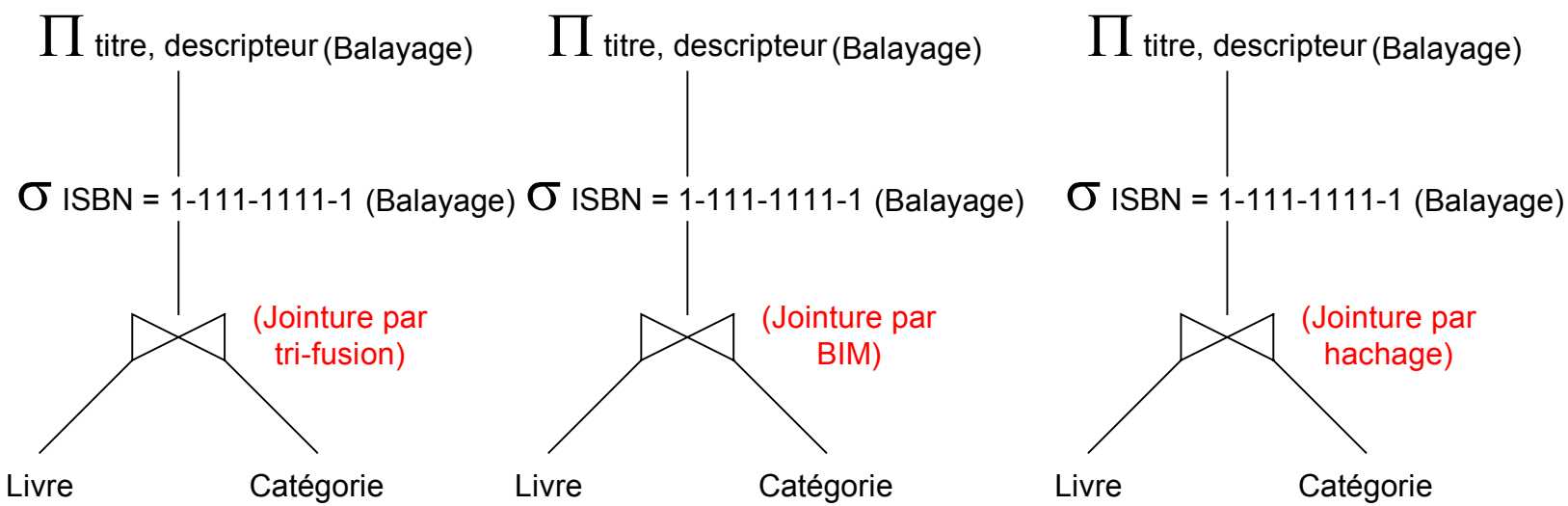
- Essentiel: pour chaque paire, au moins un fragment doit tenir en mémoire

Optimisation

- Chercher le meilleur plan d ’exécution?
  - coût excessif
- Solution approchée à un coût raisonnable
  - Générer les alternatives
    - heuristiques
  - Choisir la meilleure
    - estimation approximative du coût

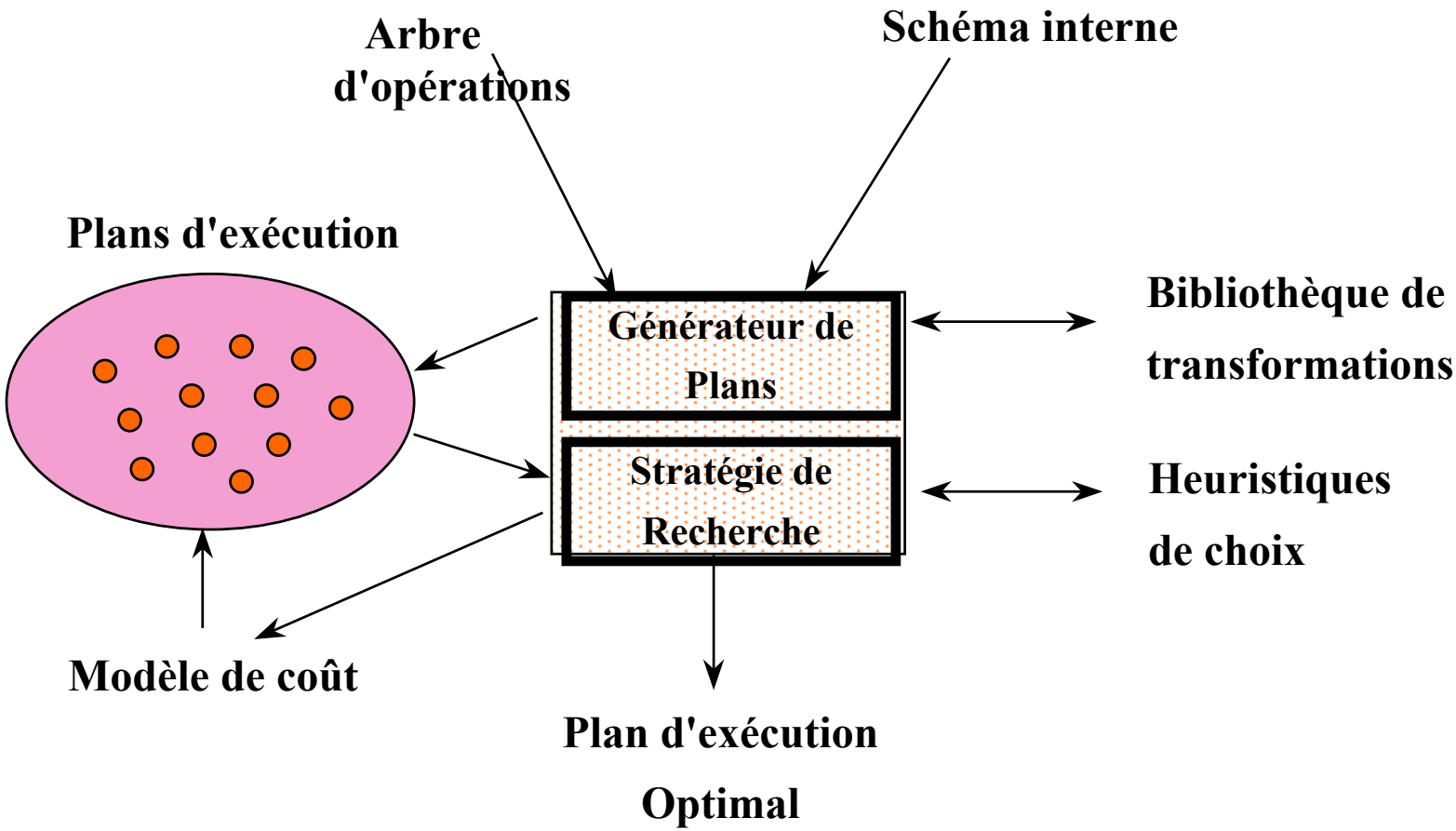
Plusieurs plans d'exécution pour un arbre algébrique

- Pour chaque opération logique
  - plusieurs choix d 'opérations physiques

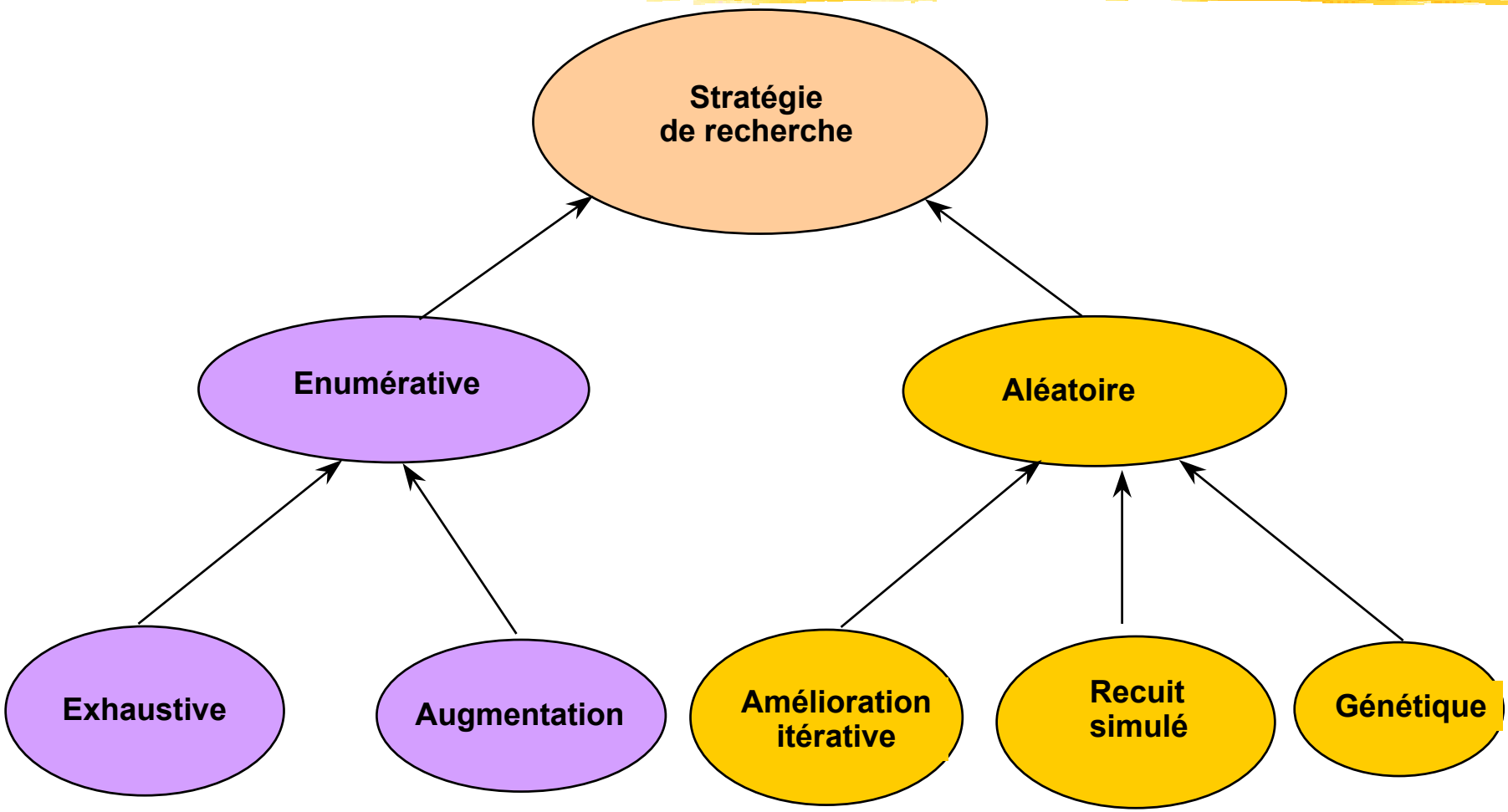


  - etc.

Choix du meilleur plan

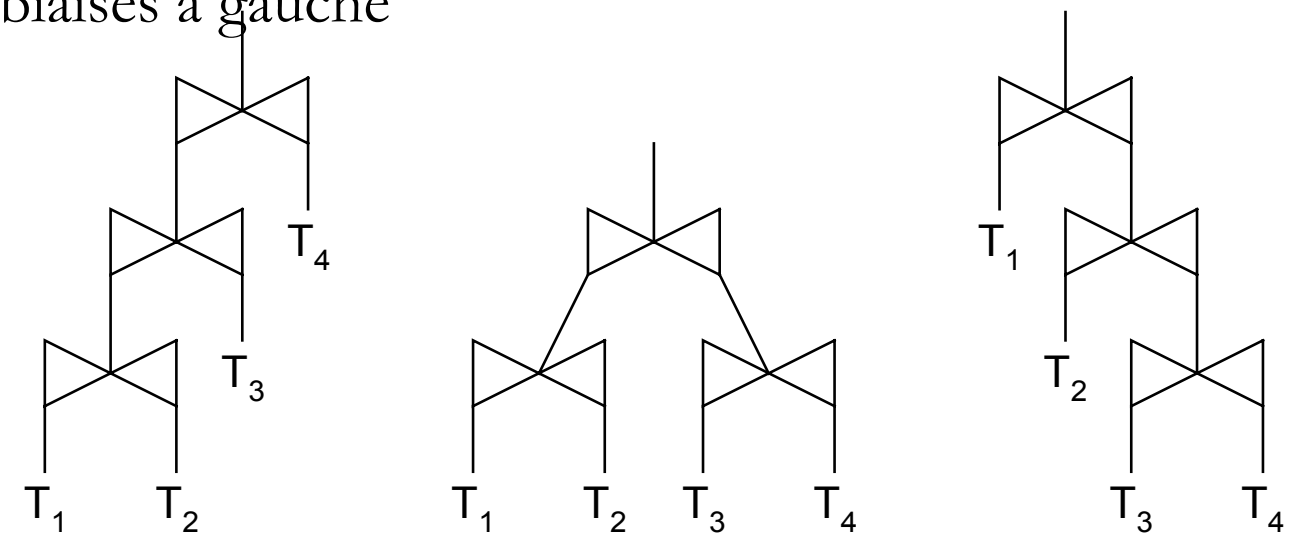


Différentes Stratégies



Heuristique : arbres biaisés à gauche seulement

- Jointure de n tables
  - (2*(n-1))!/(n-1)! ordres différents pour n tables
  - n! biaisés à gauche



Arbre biaisé à gauche | Arbre équilibré | Arbre biaisé à droite

Oracle : mode RULE

Contrôle du processus d'optimisation

- Cas Oracle
  - OUutils
    - EXPLAIN PLAN
    - SQL *Trace*
    - SQL *Analyse* (*Enterprise Manager Tuning Pack*)
    - Oracle EXPERT

Index

```
SELECT /*+ RULE*/ nom
FROM Client
WHERE noClient = 10 ;
```

```
SELECT /*+ INDEX(EMPLOYÉ INDEX_SEXE)*/ nom, adresse
FROM EMPLOYÉ WHERE SEXE = 'F'
```

Paramètre OPTIMIZER_MODE

- COST(statistique): minimise le coût estimé
  - besoin de statistiques (ANALYSE)
  - mieux mais plus cher
  - ALL_ROWS
    - minimise le temps total (plutôt MERGE JOIN)
  - FIRST_ROWS
    - minimise temps réponse (plutôt NESTED LOOPS)
- RULE (heuristique)
  - appelé à disparaître ?

Exemple d ’utilisation de EXPLAIN PLAN

SQL> start utlxplan.sql *Pour créer la table plan_table*
Table created.
...
SQL> run
  1 explain plan
  2 set statement_id = 'com'
  3 for select * from commandes, lignes_de_commande
  4 where no_commande = commande_no_commande

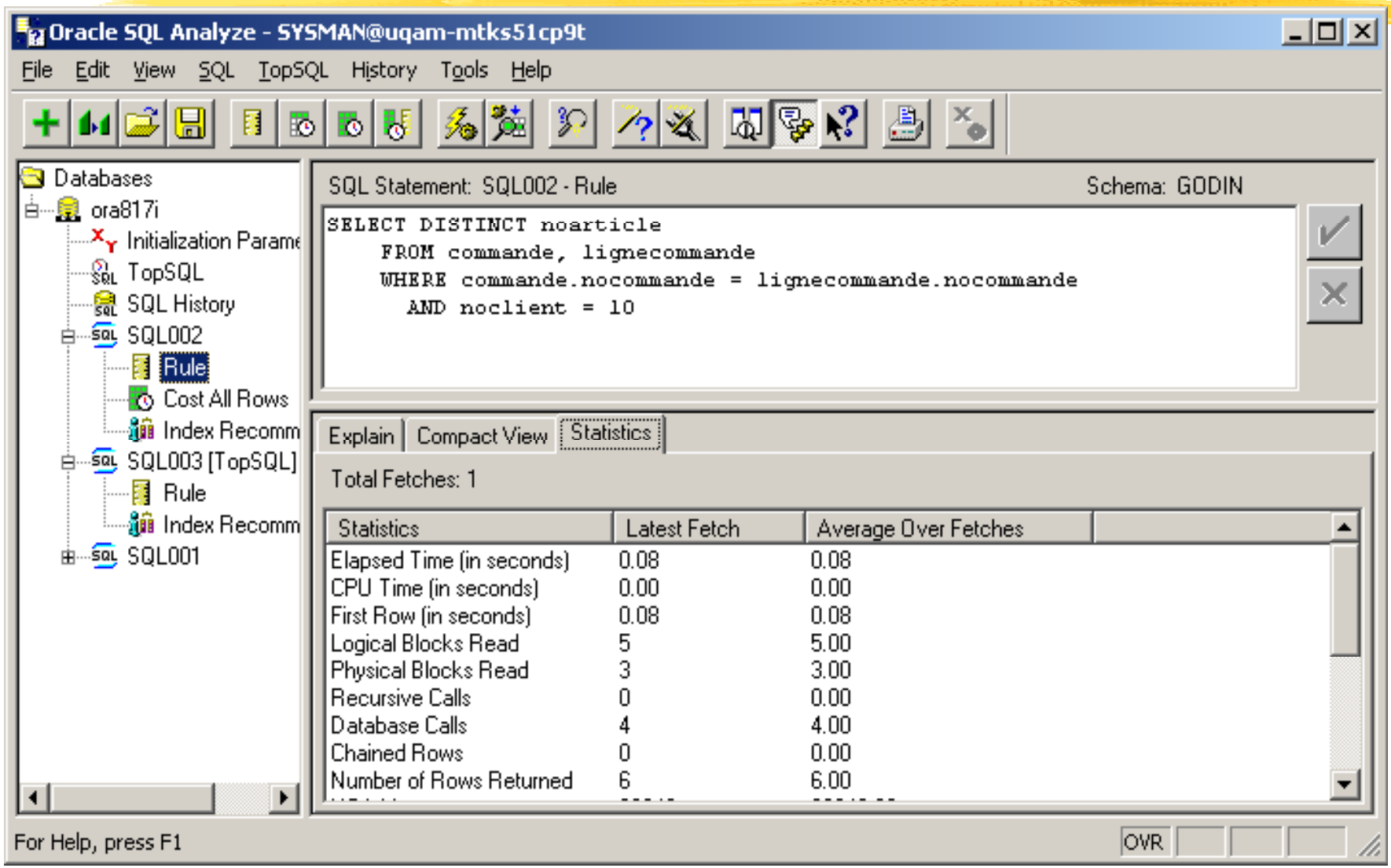
Explained.

Suite

```
SQL> run
  1  SELECT LPAD(' ',2*(LEVEL-1))||operation||' '||options
  2    ||' '||object_name
  3      ||' '||DECODE(id, 0, 'Cost = '||position) "Query Plan"
  4  FROM plan_table
  5     START WITH id = 0 AND statement_id = 'com'
  6*  CONNECT BY PRIOR id = parent_id AND statement_id ='com'

Query Plan
------------------------------------------
SELECT STATEMENT   Cost =
  NESTED LOOPS
    TABLE ACCESS FULL LIGNES_DE_COMMANDE
    TABLE ACCESS BY ROWID COMMANDES
      INDEX UNIQUE SCAN COMMANDE_PK
```

Consommation de ressources



Cas DB2

- Documentation
  - http://www-4.ibm.com/cgi-bin/software/db2www/library/pubs.d2w/report#UDBPUBS
- Paramètre OPTIMIZATION CLASS
  - Classe 0
    - algorithme vorace d'énumération des ordres de jointure
    - boucles imbriquées seulement
  - Classe 1
    - jointure par tri-fusion
  - Classe 2
    - optimisation de l 'accès par index secondaire par manipulation des listes de références
    - jointures en étoile
  - Classes 3
    - algorithme de programmation dynamique
    - réécritures des SELECT imbriqués en jointures
  - Classe 5
    - statistiques non uniformes (histogrammes)
  - Classe 7
    - examine plus de plans
  - Classe 9
    - considère toutes les alternatives

Optimisation basée sur modèle de coût

- Soit Q une requête à optimiser:
- Procédure:
  1. Énumérer tous les plans $\{P_1, \ldots, P_m\}$ pour chaque requête (notons que chaque requête possède un ensemble d'opérations $O_1, \ldots, O_k$)
  2. Pour chaque plan P_j

     Pour chaque opération O_i du plan P_j, énumérer les routines d'accès

     Sélectionner la routine ayant le coût le moins élevé

Coût(P_i) = $\Sigma_{(l=1, k)}$ min(O_l)

Coût (Q): $\Sigma_{(h=1, m)}$ Coût(P_h)

Exemple

- Une requête avec 3 opérations (une projection, deux sélections et une jointure) : P1 S1 S2 et J1
  Généralement, on exécute les sélections d'abord, ensuite la jointure et enfin la projection.
- Énumérer tous les plans:
  - Plan A: S1 S2 J1 P1 Plan B: S2 S1 J1 P1
  - Choisir un plan (commençons par Plan A)
    - Pour chaque opération, énumérer toutes les routines:

      Opération S1 : Linear Search et binary search

      Opération S2 : Linear Search et indexed search

      Opération J1 : Nested Loop join et indexed join

Exemple (suite)

- Choose the least cost access routine for each operation

  Opération S1 least cost access routine is binary search at a cost of 10 blocks
  Opération S2 least cost access routine is linear search at a cost of 20 blocks
  Opération J1 least cost access routine is indexed join at a cost of 40 blocks

- Thus the total cost for Plan A will be: 70

- Répéter le même principe pour le plan B:

  Opération S2 least cost access routine is binary search at a cost of 20 blocks
  Opération S1 least cost access routine is indexed search at a cost of 5 blocks
  Opération J1 least cost access routine is indexed join at a cost of 30 blocks

  Thus the total cost for Plan B will be: 55

  ➡ Résultat: Plan B le meilleur plan.

Hints

- Les directives "hints" sont des moyens de forcer l'optimiseur à choisir un plan d'exécution.
- Les directives peuvent influencer les plans d'exécution et par conséquent la performance des requêtes
- Exemple
  - `Sous Oracle:`
    - `SELECT /*+ some hints here */`

Hints sous Oracle

Hint	Explication
CHOOSE	If statistics are available (see DBMS_STATS package) then use Cost Based otherwise use Rule Based.
RULE	Use the Rule based Optimizer (even if table statistics are available).
ALL_ROWS	Use the Cost Based optimizer to maximize throughput (minimize system resources)
FIRST_ROWS	Use the Cost Based optimizer to minimize response time

Hints sous Oracle

Hint	Explication
ORDERED	Join tables in the order in which they appear in the FROM clause of the query. Try changing around the order to see if performance improves.
FULL (table_name)	Perform a full table scan on the named table even if an index is available.
INDEX (table_name index_name)	Forces the use of the named index on the named table (rather than a full table scan).

Exemple

```
SELECT /*+ ALL_ROWS */ nom
FROM Ouvrier;

SELECT /*+ FIRST_ROWS */ nom
FROM Ouvrier;

SELECT /*+ COST */ nom, ville
FROM Ouvrier WHERE age > 25 AND Ville in ('Poitiers', 'Niort', 'Angouleme' ,
    'LaRochelle');
```

Hints: Accès aux données

- **FULL** - Force a full table-scan. Use when querying large portions or all of a table or for small static tables.

```
SELECT /*+ FULL(<tablename>) */ * FROM <tablename>;
```

- **ROWID** - Scan a table by ROWID.
- **CLUSTER** - Cluster scan on a cluster.
- **HASH** - Cluster hash scan.
- **INDEX**:

```
SELECT /*+ INDEX(<tablename indexname [indexname]>) */
  * FROM <tablename> WHERE ...;
```

- **INDEX_COMBINE** : Forces bitmap access
- **INDEX_JOIN** - Forces the use of an index join.

Hints: Jointure

- **ORDERED** - Forces table access in a specified order from multiple selected tables

  ```
  SELECT /*+ ORDERED */ * FROM <tablename, tablename>
  ```

- **USE_MERGE** - Join each table with a sort-merge join.
- **USE_HASH** - Join each table with a hash-merge join.
- **[NO] PARALLEL** - perform full table scans in parallel by splitting a query into multiple threads running simultaneously, the results are merged to be returned as a single row set.
  - Example: This query breaks the query up into four processes.

    ```
    SELECT /*+FULL(table) PARALLEL(table, 4) */ field1, field2
    FROM table
    ```