BUT 3
Nouveaux paradigmes de bases de données
2023-2024

Gilles Nachouki

Plan du cours

- Optimisation des requêtes - App. Oracle
- Gestion des transactions - App. Oracle
- Modèle de données Relationnel-Objet - App. Oracle
- Big Data et MapReduce - App. MongoDB
- Réplication et repartition de données - App. MongoDB

L'optimisation est une étape fondamentale permettant, lors du traitement des requêtes, de réduire l'accès aux données dans la base de données et par consequent le temps total d'exécution des requêtes.

Etape 1 : Générer l'arbre algébrique de la requête

- A ce niveau, il s'agit de vérifier :
  a) la présence de certains mots-clés du langage SQL dans la requête (Select, from, where etc.);
  b) la bonne syntaxe de la requête et
  c) la validité des tables et des attributs auxquels la requête fait référence.

Cette étape génère l'arbre algébrique de la requête.

Etape 2 : Choix du plan d'exécution de la requête

- Il existe plusieurs stratégies d'exécution d'une requête : il s'agit durant cette étape de choisir la stratégie la plus avantageuse permettant de transformer une expression algébrique en une expression équivalente dans le sens où elle optimise le traitement de la requête.
  Le résultat de cette étape est un arbre algébrique 'optimisé' constituant le plan d'exécution de la requête.

Remarques : Choix du plan d'exécution de la requête

- Explorer toutes les expressions algébriques (on ne peut pas tout explorer)
- Evaluer le coût de chacune d'elles (trop long !)
- Solution : on applique des heuristiques

Etape 3 : Générer l'arbre physique

- Cette étape génère l'arbre physique de la requête exprimé sous forme d'un ensemble d'opérateurs physiques (sélection, jointure etc.).

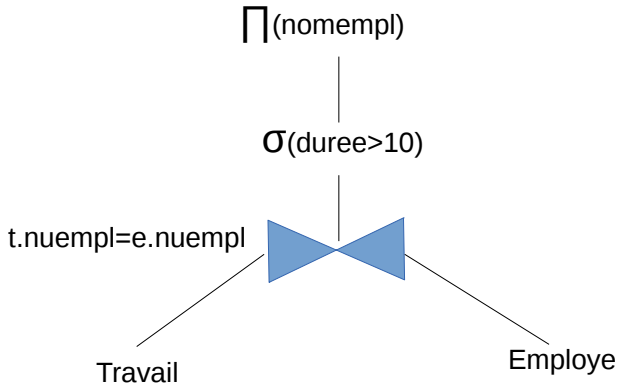
Etape 4 : Générer le plan d'exécution

- Il s'agit de générer le code correspondant à l'arbre physique ainsi obtenu et de l'exécuter sur la base de données.

La génération de l'arbre algébrique correspondant à une requête SQL est obtenue comme suit :

- Un noeud feuille est créé pour chaque relation de la requête
- Un noeud non feuille est créé pour chaque relation intermédiaire produite par une opération d'algèbre relationnelle (Selection, Projection, Jointure etc.)
- La racine de l'arbre représente le résultat de la requête
- Les opérations sont dirigées des feuilles vers la racine.

Exemple



Arbre algébrique de la requête SQL :

(Q) : Select nomempl from travail t, Employe e
Where t.nuempl = e.nuempl and duree > 10

Règles de réécriture algébrique

Le but est d'évaluer le coût de chaque réécriture afin de choisir le meilleur plan.

1- Des opérations de sélection conjonctives se décomposent en une suite d'opérations de sélection
$\sigma_{p \wedge q}(R) = \sigma_p(\sigma_q(R))$

2- Commutativité des opérations de sélection
$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$

3- Dans une séquence de projections seule la dernière opération est nécessaire
$\prod_a(\prod_b \ldots(R)) = \prod_a(R)$

4- Substituer la sélection et la projection
$\prod_a(\sigma_p(R)) = \sigma_p(\prod_a(R))$: si la condition de sélection p n'implique que les attributs de la liste de projection a.

Règles de réécriture algébrique - suite

5- Commutativité de la jointure (et de X)
$R \rhd\lhd_{R.a=S.b} S = S \rhd\lhd_{S.b=R.a} R$
$RXS = SXR$

6- Substituer la sélection et la jointure (ou le produit cartésien)
$\sigma_p(R \rhd \lhd_{R.a=S.b} S) = \sigma_p(R) \rhd \lhd_{R.a=S.b} S$: si tous les attributs de la condition n'impliquent que des attributs d'une des relations concernées par la jointure (par exemple R);
Si la condition p peut être écrite sous la forme p1 et p2 où p1 n'implique que les attributs de R et p2 n'implique que les attributs de S alors les deux opérations peuvent commutées de la manière suivante : $\sigma_p(R \rhd \lhd_{R.a=S.b} S) = \sigma_{p1}(R) \rhd \lhd_{\sigma_{p2}}(S)$

7- Substituer la projection et la jointure (ou le produit cartésien)
$\prod_l(R \rhd \lhd_{R.a=S.b} S) = (\prod_{l1}(R)) \rhd \lhd_{R.a=S.b}(\prod_{l2}(S))$: on suppose que la liste de projection l sont des attributs de R (l1) et de S (l2) et que la condition de jointure n'implique que des attributs dans l.

Règles de réécriture algébrique - suite

8- Commutativité de l'union et de l'intersection
$R \bigcup S = S \bigcup R$

9- Substituer la sélection avec les opérations sur des ensembles
$\sigma_p(R \bigcup S) = \sigma_p(R) \bigcup \sigma_p(S)$

10- Substituer la projection et l'union
$\prod_c(R \bigcup S) = \prod_c(R) \bigcup \prod_c(S)$

11- Associativité de la jointure
$(R \rhd \lhd_{R.a=S.b} S) \rhd \lhd_{S.b=T.c} T = R \rhd \lhd_{R.a=S.b} (S \rhd \lhd_{S.b=T.c} T)$

12- Substituer la séquence d'opérations : $\sigma_p(\text{RXS}) = R \rhd \lhd_p S$

Des heuristiques pour l'application des règles de réécriture

1- Appliquer les opérations de sélection le plus tôt possible

2- Appliquer les opérations de projection le plus tôt possible

3- Combiner le produit cartésien avec une opération de sélection dont le prédicat représente une condition de jointure

4 Utiliser l'associativité (commutativité) pour réorganiser les feuilles afin que les opérations de sélection les plus restrictives s'exécutent en premier.

D'autres critères sont nécessaires

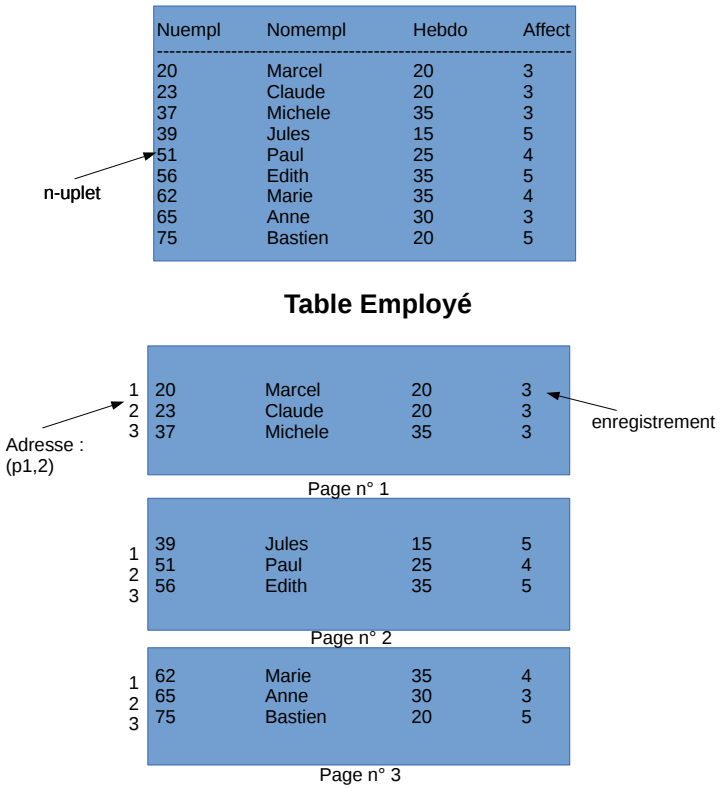
Les réécritures algébriques des requêtes SQL ne sont pas suffisantes durant la phase d'optimisation. D'autres critères s'ajoutent durant cette phase :

1- L'accès aux données (accès séquentiel ou indexé etc.)

2- Les algorithmes qui sont utilisés lors des opérations de sélection, projection ou de jointure.

3- Des statistiques : nombre de n-uplets dans les tables, sélectivité des attributs etc.

Stockage physique des tables



Stockage sur disque de la table Employé
Sur trois pages de données

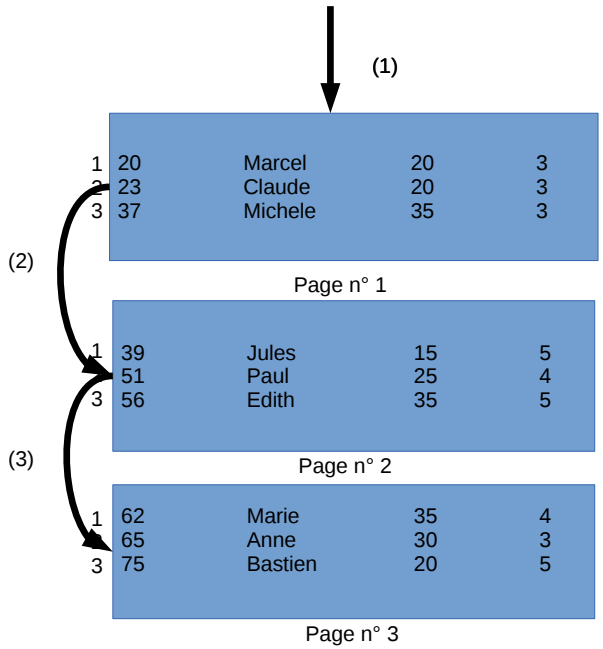
Accès aux données

1. Accès séquentiel
2. Accès par index (primaire et secondaire)
3. Accès par table de hachage
4. D'autres méthodes d'accès (B-arbre etc.) [1]

[1] Ces méthodes ne sont pas abordées ici

Accès séquentiel

Dans ce type d'accès les enregistrements physiques dans les fichiers sont lus séquentiellement les uns après les autres.



Accès par index

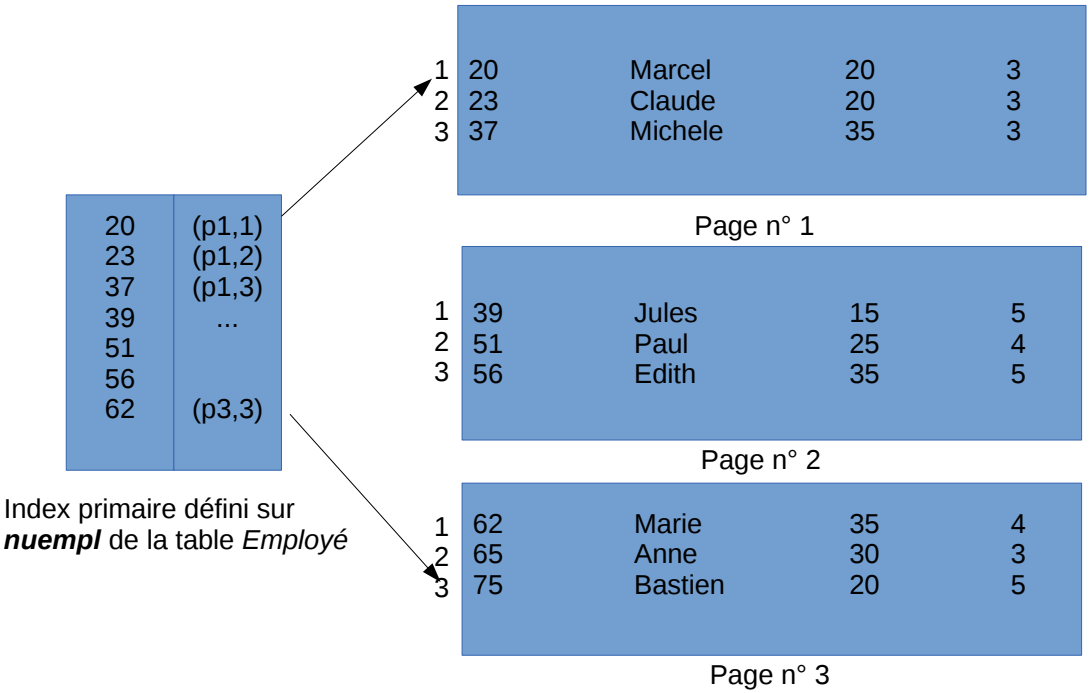
Un index est une structure de données qui permet à un SGBD d'obtenir plus rapidement les n-uplets afin d'améliorer le temps de réponse. Il existe plusieurs types d'index dont les principaux sont :

- index primaire
- index secondaire

Accès par index

Index primaire

Les n-uplets dans les pages sont triés de manière séquentielle selon les valeurs de la clé primaire. L'index primaire est basé sur cette clé qui garantit une valeur unique de chaque tuple de la table. L'index est trié selon les valeurs de la clé primaire.

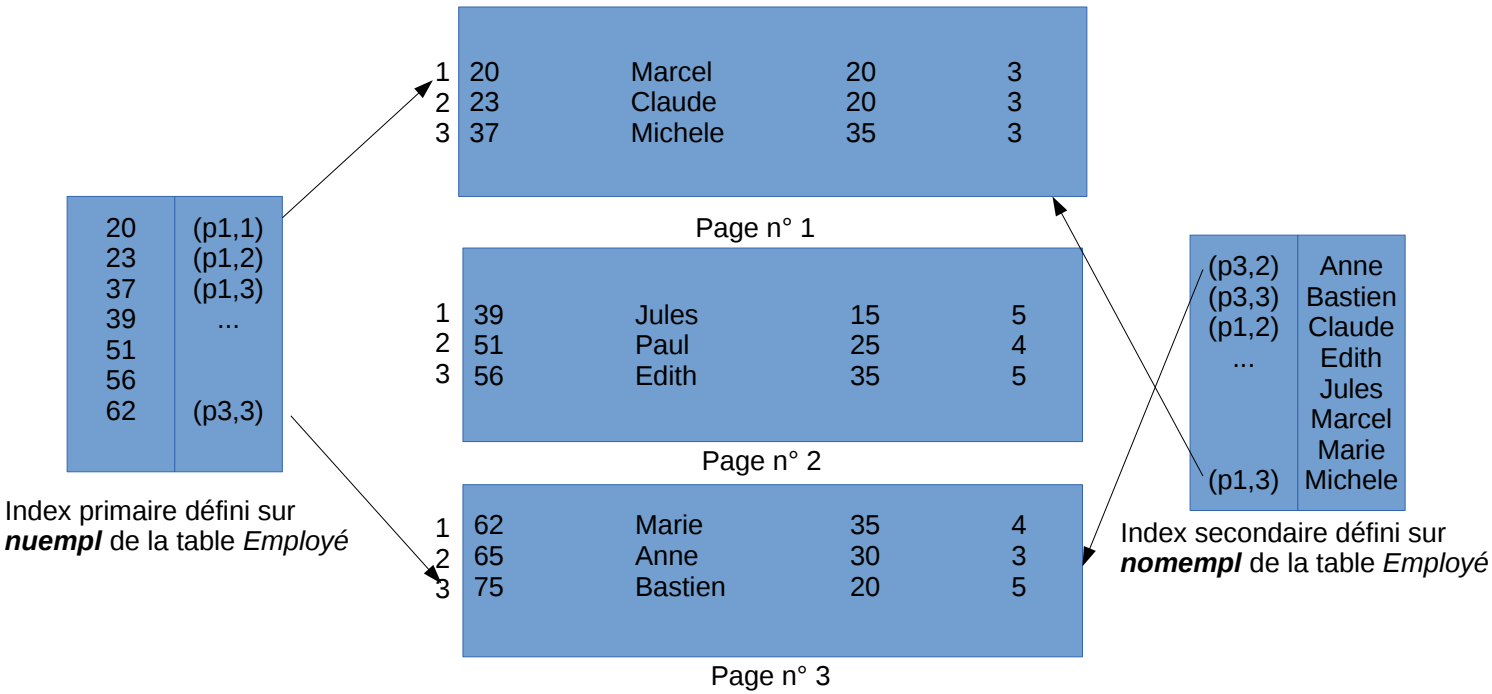


Index primaire défini sur ***nuempl*** de la table *Employé*

Accès par index

Index secondaire :
L'index secondaire est construit sur un attribut qui est souvent utilisé dans les requêtes. Cet index ne demande aucun tri des pages de données.
Une table peut avoir au plus un index primaire mais peut posséder plusieurs index secondaires.

Accès par index



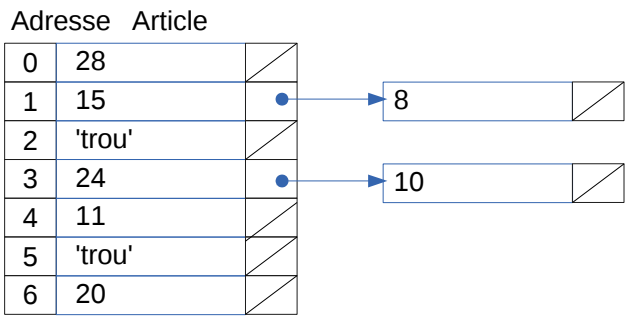
Accès par hachage

Une fonction de hachage (hash-coding) est une fonction f qui, à toute clé c de l'ensemble des clés, associe un (et un seul) numéro ou plutôt une adresse et une seule parmi l'ensemble des adresses disponibles [1..N] : a = f(c).

Exemple

Adresse	Article
0	28
1	15
2	'trou'
3	24
4	11
5	'trou'
6	20

Si l'on ajoute les articles de clés 8 et 10, tous les deux entrent Collision avec les articles 15 et 24.



Avantage d'une structure de hachage

- la structure n'occupe aucun espace disque contrairement à d'autres index
- elle permet un accès direct aux données.

Algorithmes de jointure

L'opération de jointure est une opération couteuse. Nous étudions ici quelques algorithmes qui implémentent l'opérateur algébrique de jointure :

- Boucles imbriquées sans index (Nested-loop Join)
- Boucles imbriquées avec index (Single-loop Join)
- Tri-Fusion (Sort-merge Join)
- Hachage (jointure par hachage)

Nested-loop Join ou Boucles imbriquées sans index

Require: : Table R et S
Ensure: : Jointure T entre R et S avec la condition : R.a=S.b
T = Φ
Pour chaque tuple r dans R faire
Pour chaque tuple s dans S faire
If (r.a = s.b) then $T = T \cup \{r \rhd\lhd s\}$
endif
Fin-faire
Fin-faire

Single-loop Join ou Boucles imbriquées avec index

Principe : Pour chaque n-uplet r de R on utilise l'index de S pour rechercher le(s) n-uplet(s) s correspondant de S. Ensuite, on garde seulement les couples de n-uplets (r,s) qui ont la même valeur de jointure.

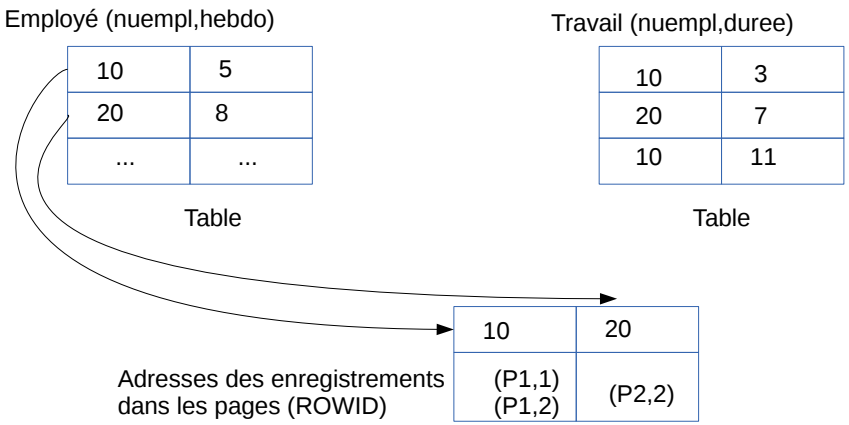
Single-loop Join ou Boucles imbriquées avec index

Require: : Table R, S et un index Ind défini sur S.b
Ensure: : Jointure T entre R et S avec la condition : R.a=S.b

$T = \Phi$
Pour chaque tuple r dans R faire
Pour chaque tuple $s \in Ind(r.a)$ faire
$T = T \cup \{r \rhd\lhd s\}$
Fin-faire
Fin-faire

La fonction Ind(r.a) renvoie une liste d'adresse (ou des ROWID) concernant les tuples dans S vérifiant la condition de jointure.

Exemple



*Index défini sur nuempl dans la table **Travail***

Avec la condition hebdo < duree
le résultat de la jointure entre les deux
Tables est la suivante : 10 ; 5 ; 10 ; 11

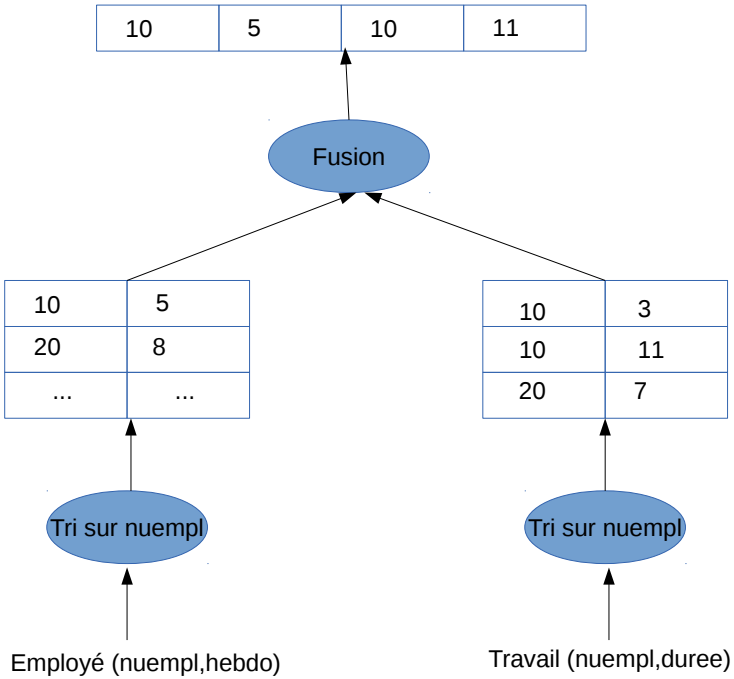
Sort-merge Join ou Jointure par tri-fusion

Require: : Table R et S
Ensure: : Jointure T entre R et S sous la condition : R.a=S.b
T = Φ
Tr = trier(R/A), trier la table R sur l'attribut A
Ts = trier(S/B), trier la table S sur l'attribut B
T = Fusion(R,S)

La *fusion* consiste à parcourir en même temps les deux relations Tr et Ts et à garder les tuples qui vérifient la condition Tr.a = Ts.b

Exemple : Tri- fusion

Condition : Employé.nuempl=Travail.nuempl et hebdo < duree



Jointure par hachage

Require: : Table R, S et H (fonction de hachage)
Ensure: : Jointure T entre R et S sous la condition : R.a=S.b

T = Φ
Pour chaque tuple r dans R faire
Placer r à l'adresse p=H(R.a)
Pour chaque tuple s dans S faire
lire l'adresse p retournée par H(S.b)
s'il existe un tuple r ∈ p alors
si r.a = s.b alors T = T ∪ {r ⋈ s }
Fin-Si
Fin-Faire(s)

Algorithmes implémentant l'opérateur algébrique de sélection

Nous distinguons quelques algorithmes implémentant cet opérateur :

- Recherche linéaire : Pour chaque tuple on regarde si le tuple vérifie la condition de sélection dans un fichier non trié et sans index.
- Index primaire : il s'agit d'utiliser un index primaire défini sur un attribut clé de la table et de vérifier si la condition est vérifiée.
- Index secondaire : il s'agit d'utiliser un index secondaire défini sur un attribut non clé de la table.

Algorithmes implémentant l'opérateur algébrique de projection

L'opérateur algébrique de projection nécessite essentiellement les deux étapes suivantes :

- Supprimer les attributs inutiles.
- Eliminer les n-uplets en double. Une approche consiste à trier les tuples dans la table réduite. Le résultat du tri permet de rassembler les n-uplets en double pour faciliter leur élimination. Remarque: lorsque l'opération de jointure renferme un attribut clé, aucune élimination de doublons n'est nécessaire.

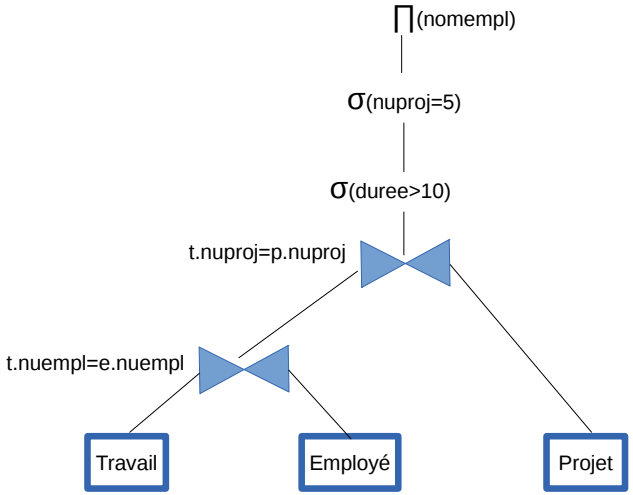
Statistiques et Choix du meilleur plan

- Le nombre des tuples dans les tables R et S détermine l'algorithme à utiliser lors d'une opération de jointure entre deux relations R et S.
- La sélectivité d'un attribut S.b est le rapport entre le nombre de valeurs distinctes de B et la cardinalité de S. Par exemple, cette information détermine la nécessité d'utiliser un index lors du traitement d'une requête portant sur l'attribut B
- Calcul du coût des différentes opérations de sélection, jointure et projection.

Etape 3 : Passage de l'arbre algébrique vers l'arbre physique

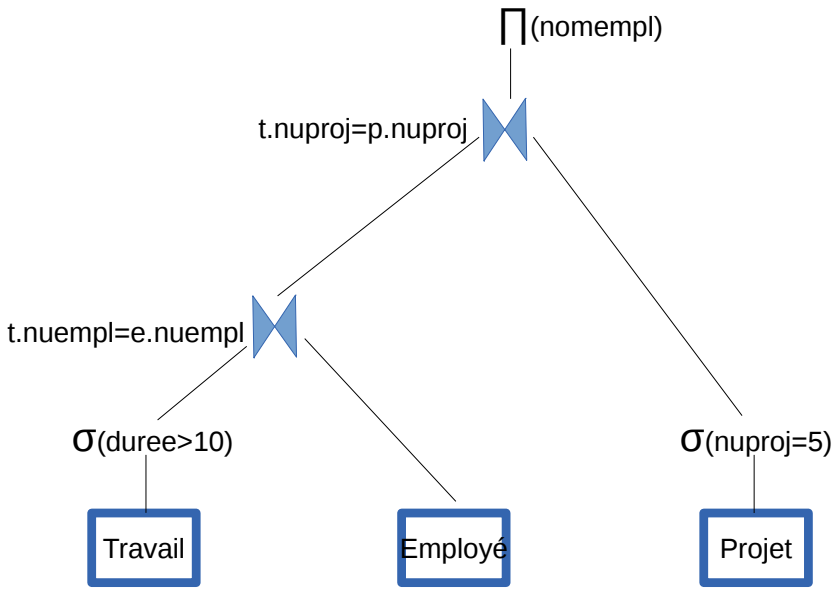
Exemple : Les employés qui travaillent plus de dix heures sur le projet numéro 5.

Requête SQL (Q): select nomempl from employe e, travail t, projet p where t.duree > 10 and p.nuproj = 5 and p.nuproj = t.nuproj and t.nuempl = e.nuempl.



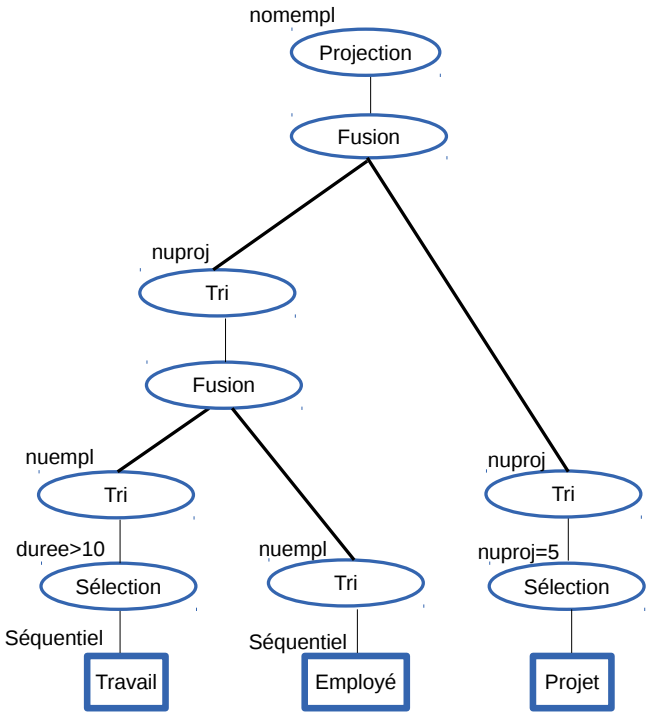
Arbre algébrique possible de la requête

Arbre algébrique optimisé



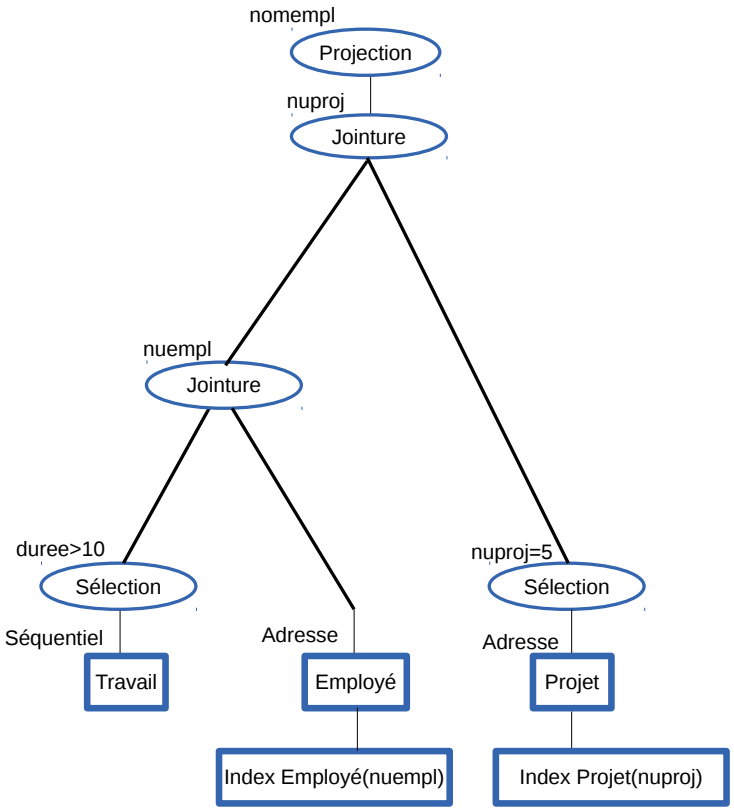
Arbre algébrique optimisé de la requête

Arbre physique sans index



Plan physique de la requête (sans index)

Arbre hysique avec index



Plan physique de la requête (avec index)

- Le plan d'exécution est l'expression par le SGBD de la méthode d'accès aux données pour répondre à une requête
- Oracle génère plusieurs plans d'exécution pour une requête. Afin de choisir un plan optimisé de la requête, Oracle estime le coût de ces plans.

Opérations d'accès aux tables

- Full Table Scan : La table dans cette opération est lue entièrement
- Table Access By *RowID* (ou par adresse) : Cette opération permet un accès direct à un enregistrement de la table en utilisant des identifiants *RowID*. Ce dernier contient un numéro du bloc et un déplacement à l'intérieur du bloc afin d'accéder au n-uplet.

Opérations d'accès aux index

- Unique Scan : Parcourt l'index pour localiser une clé unique (exemple : nuempl=17)
- Range Scan : Parcourt une partie de l'index de manière ordonnée (exemple : durée between 10 and 25)
- Index Full Scan : Parcourt l'index entièrement dans l'ordre de l'index

Opérations de jointure

- Nested Loop (boucles imbriquées) : Parcourt les sous_ensembles pour chaque valeur dans l'ensemble de départ
- Merge Join : Permet le rapprochement de deux ensembles triés
- Sort Join : Permet de trier un ensemble de n-uplets
- Hash join : Permet de construire une table de hachage pour accéder aux valeurs de deux n-uplets

Explain plan for

Pour obtenir le plan d'exécution d'une requête SQL il faut lancer la commande EXPLAIN PLAN FOR ordre sql; Cette commande analyse la requête et stocke le résultat dans une table d'audit. Pour voir le plan de la requête on peut lancer les commandes suivantes :

select * from plan_table ou

select * from table (dbms_xplan.display);

Exemple

(Q) : select nomempl from employe e, travail t, projet p where t.duree > 10 and p.nuproj = 237 and p.nuproj = t.nuproj and t.nuempl = e.nuempl;

SQL | 2,462 secondes

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		4	10
HASH JOIN		4	10
Access Predicates			
P.NUPROJ=T.NUPROJ			
MERGE JOIN		4	6
TABLE ACCESS (BY INDEX ROWID)	EMPLOYE	20	2
INDEX (FULL SCAN)	PKNUEMPL	20	1
SORT (JOIN)		4	4
Access Predicates			
T.NUEMPL=E.NUEMPL			
Filter Predicates			
T.NUEMPL=E.NUEMPL			
TABLE ACCESS (FULL)	TRAVAIL	4	3
Filter Predicates			
AND			
T.NUPROJ=237			
T.DUREE>10			
TABLE ACCESS (FULL)	PROJET	1	3
Filter Predicates			
P.NUPROJ=237			