Optimisation de Bases de Données

Nicolas Travers

CNAM Paris
nicolas.travers (at) cnam.fr

Plan

Qu'est-ce qu'une base de données

- Collection de données : Relation / Table
  - **homogènes**
  - **structure précise**
- Permet l'interrogation : Algèbre / SQL
  - Des données
  - Grâce à la structure
- Différent des recherches par mot-clés sur le web

SGBD

- **S**ystème de **G**estion de **B**ase de **D**onnées
- Permet pour les données :
  - Stockage
  - Interrogation
  - Mise à jour
  - Partage / Concurrence d'accès

Modèle Relationnel

Indépendance Structure / Langage

- Langage déclaratif : SQL
- Vue ensembliste des données
- Déclare ce qu'il faut faire
- Ne dit pas comment le faire

Indépendance

- **Niveau Logique** : utilisateur
  - BD manipulée à l'aide d'un langage de requêtes - SQL
  - Vision *abstraite* du niveau physique
- **Niveau Physique** : stockage, implémentation du langage, multi-plateformes
  - Stockage différents sur différentes plateformes
  - Implémentation du langage et traduction vers le langage machine
  - Modifications physiques n'ont pas d'impact sur le langage
- **Langage Déclaratif**
  - Déclare ce qu'il faut faire
  - Ne dit pas comment le faire

Structures

- Type unique : **Relation** (*Table*)
  - Ensemble de *n-uplets* (*tuples*)
  - Attributs définis des types simples (atomiques)
- Structure de la base : **Schéma**
  - Ensemble de schémas de relations
  - Schéma d'une relation : **Nom et liste d'attributs**
  - Ex : Cours(Date, Heure, Prof, NbAuditeur, Intitule)

Exemple de Relation

Table **Cours** :

Date	Heure	Prof	NbAuditeur	Intitule
2017-02-28	18h15	Travers	30	Introduction
2017-03-15	18h15	Travers	32	Indexes
2017-03-22	18h30	Travers	28	Optimisation
2017-03-29	18h00	Hamdi	27	Dénormalisation
2017-05-15	18h15	Travers	22	Tuning

Algèbre Relationnelle

- Relations manipulées par des Opérations
- **Opérateur algèbrique**
  - Une ou plusieurs relations en entrée
  - Une et une seule relation en sortie
- Requête algébrique
  - Ensemble d'opérateurs
  - Forme un arbre à une ou plusieurs branches

Opérations Algébriques

- Sélection(σ) : Restriction sur valeur des tuples
- Projection(π) : Suppression de colonnes (allège le résultat)
- Jointure($\bowtie$) : Fusionne deux ensembles sur valeur
- Différence($-$) : Sous ensemble non présent dans le second
- Union($\cup$) : Union des deux ensembles
- Exemples d'expression :
  - $\pi_{Date}(\sigma_{Intitule='Introduction'}(Cours))$
  - $\pi_{Intitule}(\sigma_{Prof='Travers'}(Cours))$
  - $\pi_{Codes}((\sigma_{Prof='Travers'}(Cours)) \bowtie (Codes\_Cours)$

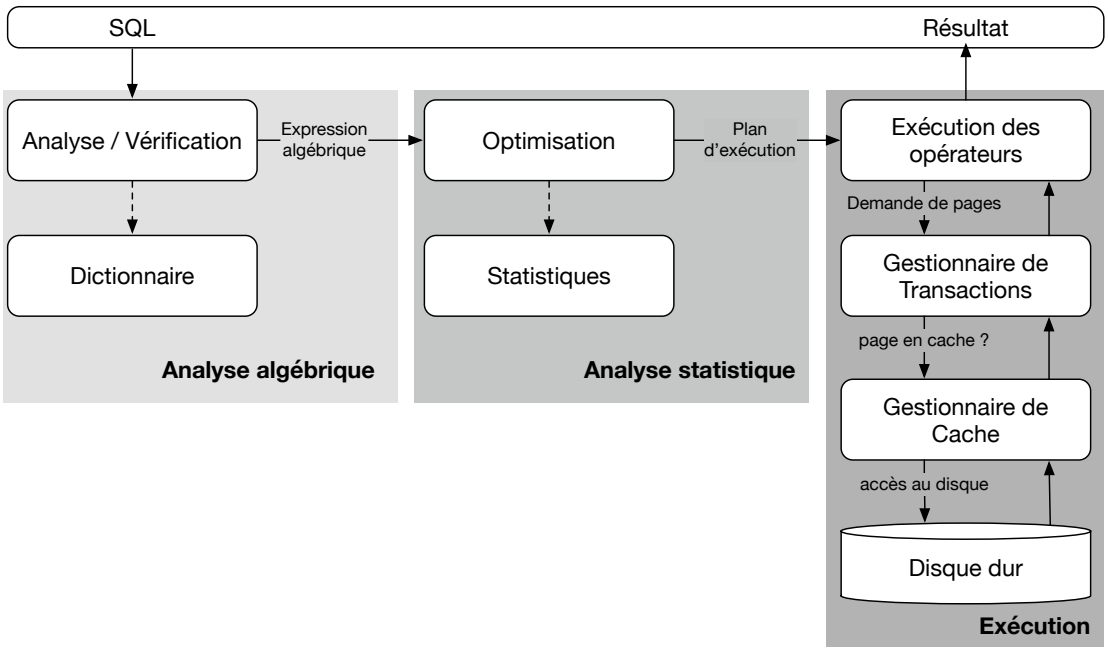
SQL

- Standard d'interrogation (régulièrement étendu)
- Clauses de base :
  - **SELECT** : projection des résultats et agrégats
  - **FROM** : relation nécessaire à l'évaluation
  - **WHERE** : sélections sur les attributs
- Plus proche du Calcul Relationnel que de l'algèbre (cf Cours)

Pouvoir d'expression

- Le noyau a autant d'expressivité que l'algèbre
  - Pour toute requête SQL (avec la base)
  - On trouve son équivalent en relationnel
- Exemples d'expression :
  - $\pi_{Date}(\sigma_{Intitule='Introduction'}(Cours))$
    SELECT date FROM Cours WHERE Intitule='Introduction'
  - $\pi_{Intitule}(\sigma_{Prof='Travers'}(Cours))$
    SELECT Intitule FROM Cours WHERE Prof='Travers'
  - $\pi_{Code}((\sigma_{Prof='Travers'}(Cours)) \bowtie (Code\_Cours)$
    SELECT Code FROM Cours C, Code_Cours CC WHERE Prof='Travers'
    AND C.IDPROF=CC.IDPROF

Cycle de vie d'une requête SQL

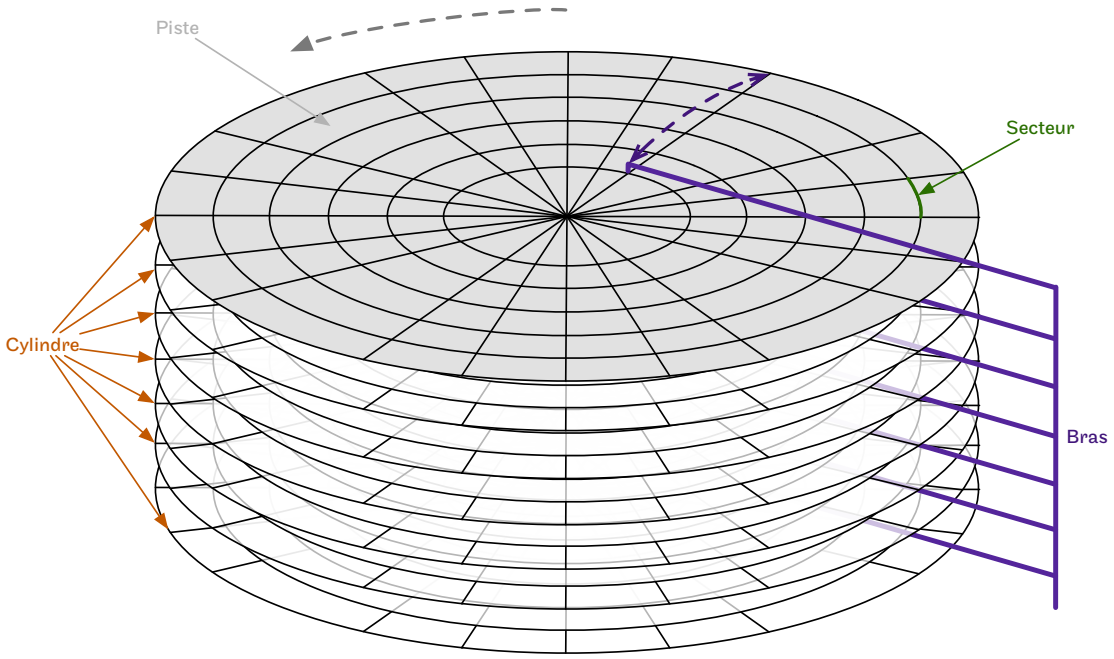


Qu'est-ce que l'optimisation dans un SGBD

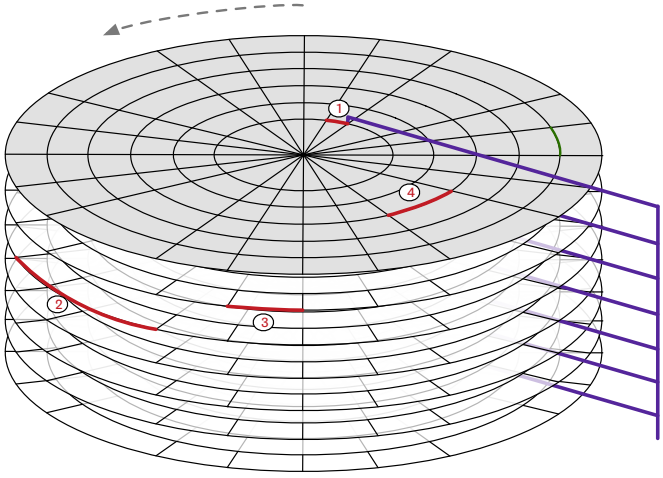
- Requête ⇒ Arbre d'opérateurs (algèbre relationnelle)
- Opérateur ⇒ Accès aux données
  - Coût en mémoire (place prise en mémoire)
  - Coût en accès aux données (I/O coûte cher)

⇒ **Minimiser les accès au disque**

Disque dur



Disque dur : Lecture d'un disque



- Délai rotationnel (rpm)

rpm	latency (ms)
15 000	2
10 000	3
7 200	4,16
5 400	5,55
4 800	6,25

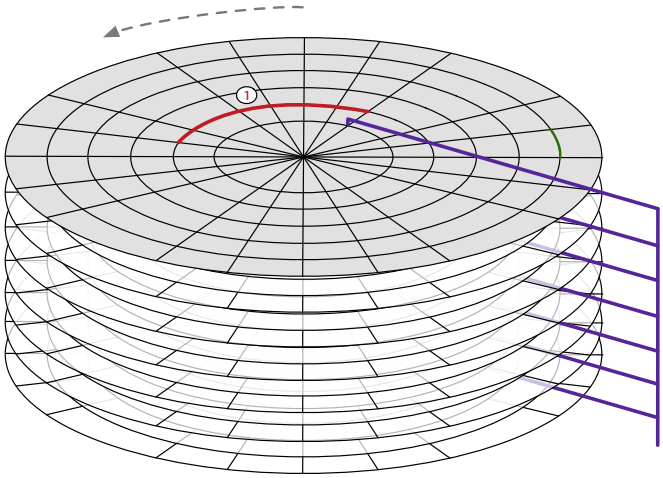
- Temps latence $\sim$ 9ms

  Dépend de :
  - Changement de piste

- Taux de transfert :
  - 125 Mo/s
  - 300 Mo/s en SATA (buffer)
- Coût : 0.04$/Go

Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

Disque dur : Lecture d'un disque



- Délai rotationnel (rpm)

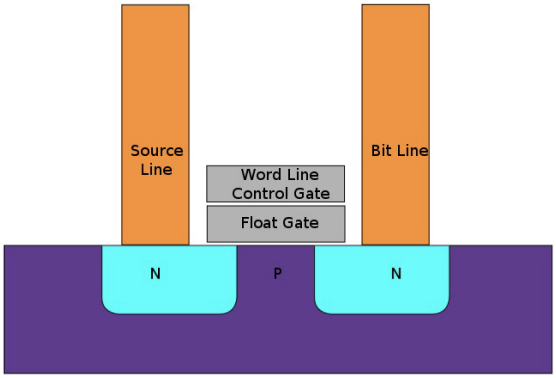
rpm	latency (ms)
15 000	2
10 000	3
7 200	4,16
5 400	5,55
4 800	6,25

- Temps latence $\sim$ 9ms

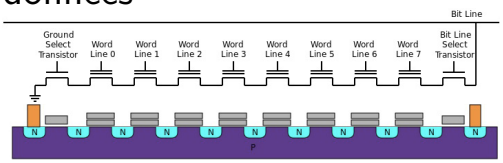
  Dépend de :
  - Changement de piste
  - Fragmentation (secteurs)
- Taux de transfert :
  - 125 Mo/s
  - 300 Mo/s en SATA (buffer)
- Coût : 0.04$/Go

Disque SSD : **Solid State Drive**

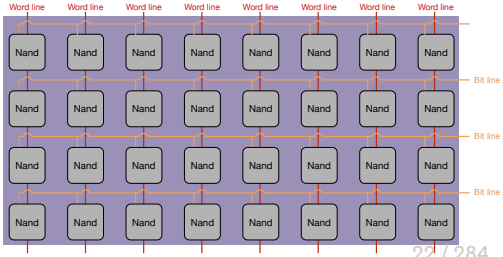
- Mémoire flash de type `Nand`
  - *Floating gate* (1nm)
  - Capture un électron (besoin forte intensité)



- 1 lecture de porte par ligne
  → Fragmentation des données



- Ecritures = Interférences
  → Ecriture par blocs entiers



Disque SSD : vitesse & durée de vie

- Types de *floating gates*
  - `SLC` : Single level cell - 1 bit
  - `MLC` : Multi-level Cell - 2 bits
  - `TLC` : Triple-level Cell - 3 bits
- Distribution des écritures
  - A terme, des secteurs corrompus
  - Problèmes de pérénité

Type	Nb cycles d'écritures
SLC	10k à 20k
MLC	3k à 5k
TLC	1k

- Taux de transfert : 130 - 3000 Mo/s
- Coût : 0.50$/Go

Entrées/Sorties

- Une entrée-sortie (I/O) $\Rightarrow$ Page disque
  - En anglais : *Block*
  - $\simeq$ Secteur sur le disque
  - $=$ 8 Ko (par défaut sous Oracle)
- Un fichier de données : plusieurs *pages* disques
- Plusieurs tuples par pages (dépend de la taille d'un tuple)

Exemple de stockage de *Cours*

CODE	Intitule	Responsable	Élèves
NFE106	Ingénierie et optimisation de Base de Données	Nicolas Travers	30
NFP107	Initiation à la Base de Données	Michel Crucianu	100
NFE204	Bases de données documentaires et distribuées	Philippe Rigaux	50
NFE205	Base de Données Avancées	Michel Crucianu	20
NFE156	Pratiques et Outils de DBA	Nicolas Travers	15
NFA011	Dev. d'appli. pour les bases de données	Marin Ferecatu	40
NFE102	Infra. Techno. pour le Commerce Electronique	Philippe Rigaux	20
NSY135	Patrons, Framework, ORM	Philippe Rigaux	30
NFE209	SI - Audit et Gouvernance	Elisabeth Métais	25

Exemple de stockage de *Cours*

Page	CODE	Intitule	Responsable	Élèves
P1	NFE106	Ingénierie et optimisation de Base de Données	Nicolas Travers	30
	NFP107	Initiation à la Base de Données	Michel Crucianu	100
	NFE204	Bases de données documentaires et distribuées	Philippe Rigaux	50
P2	NFE205	Base de Données Avancées	Michel Crucianu	20
	NFE156	Pratiques et Outils de DBA	Nicolas Travers	15
	NFA011	Dev. d'appli. pour les bases de données	Marin Ferecatu	40
P3	NFE102	Infra. Techno. pour le Commerce Electronique	Philippe Rigaux	20
	NSY135	Patrons, Framework, ORM	Philippe Rigaux	30
	NFE209	SI - Audit et Gouvernance	Elisabeth Métais	25

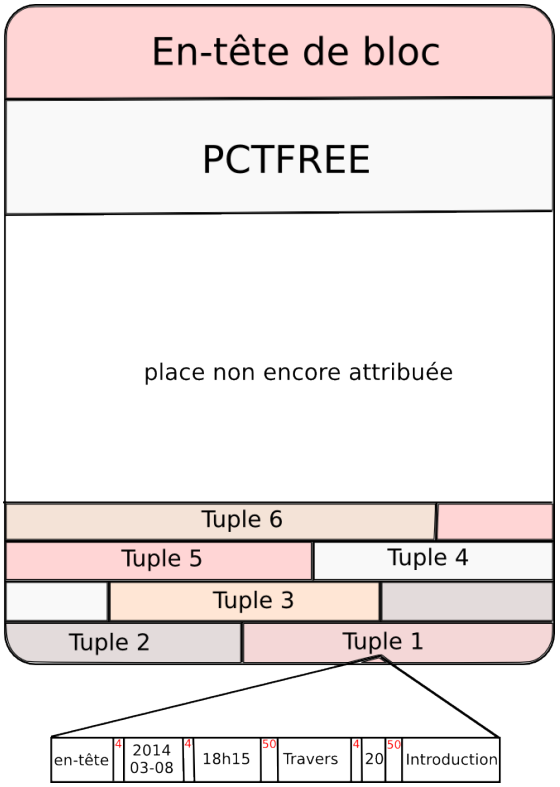
Page dique

Une page [1] contient :

- En-tête [2] :
  - Type de données (Relation, index, cluster...)
  - Schéma de données (attributs, null, blob...)
  - Pointeurs sur places vides, page suivante et précédente
- Place vide pour mises à jour
  - PCTFREE [3] / PCTUSED
- Les données
  - Tuples en entier
  - En-tête de tuple [4] (identification, valeurs NULL, verrous)
  - À chaque attribut s'associe sa taille [5]

1. Par défaut 8 ko : 8192 o
2. Entre 100 et 200 octets. Dans ce cours, nous utiliserons **192 o**
3. Par défaut 10%
4. Variable, mais en moyenne 3 octets
5. Taille de colonne entre 1 et 3 octets. En moyenne 1 octet

Pages et Taille d'une table



- Cours(Date, heure, nom, nb, intitule)
  5000 tuples
  Nb, Dates et heures : 4 o
  Texte : 50 o en moyenne.
- *Block* : 8 ko $\Rightarrow$ 8192 o
- $|DataBlock| = (8192 - 192) \times \frac{100 - PCTFREE}{100} =$ **7200 o**
- $|Tuple|$: $3 + 3 \times (4 + 1) + 2 \times (50 + 1) =$ **120 o**
- Nb tuples par page : $\left\lfloor \frac{7200}{120} \right\rfloor = 60 t/p$
- Nb pages : $\left\lceil \frac{5000}{60} \right\rceil = 84 pages$

Types de données

Types numériques

Type	Taille	Interval de valeurs
TINYINT	1 octet	0 à 255 ou -128 à 127
SMALLINT	2 octets	-32768 à 32767 ou 0 à 65535
MEDIUMINT	3 octets	-8388608 à 8388607 ou 0 à 16777215
INT, INTEGER	4 octets	-2147483648 à 2147483647 ou 0 à 4294967295
BIGINT	8 octets	$\sim 10^{19}$
FLOAT(p)	4 octets if 0≤p≤24, 8 octets if 25≤p≤53	nombre de décimales
FLOAT	8 octets	Par defaut, 53 décimales
DOUBLE(p), REAL	8 octets	
DECIMAL(M,D), NUM(M,D)	Variable	
BIT(M)	$\sim (M + 7)/8$ octets	

Types Dates et Heures

Type	Taille
DATE	3 octets
TIME	3 octets
DATETIME	8 octets
TIMESTAMP	4 octets
YEAR	1 octet

Types textuel

Type	Taille
CHAR(M)	M octets ou ou M x 2 octets, 0≤M≤255
BINARY(M)	M octets, 0≤M≤255
VARCHAR(M)	M + 1o (0-255 o), M + 2o (> 255 o)
BLOB, TEXT	L + 2 octets, where L≤215
ENUM('v1','v2',...)	1o (255 valeurs), 2o (65 535 valeurs)
SET('v1','v2',...)	1, 2, 3, 4, or 8 octets (64 valeurs maximum)

Exemple simple d'accès

- $\pi_{Date}(\sigma_{Intitule='Introduction'}(Cours))$
  SELECT date FROM Cours WHERE Intitule='Introduction'
- Opérateur de sélection nécessaire.
  - Si pas d'index sur l'attribut *Intitule* :
  - Parcours chaque page (physique) de la relation *Cours* : `TABLE ACCESS FULL`
  - Si le tuple contient 'Introduction', le tuple est projeté
- **Coût de l'opération :** Nombre de pages de *Cours*

Notations

- **Tables :**
  - $|\mathcal{R}|$: Nb de pages de la relation $\mathcal{R}$
  - $|\mathcal{R}'|$: Nb de pages de la relation $\mathcal{R}$ après modification
  - $||\mathcal{R}||$: Nb de n-uplets de la relation $\mathcal{R}$
  - $||\mathcal{R}'||$: Nb de n-uplets de la relation $\mathcal{R}$ après sélection
- **Index :**
  - $|\mathcal{I}|$: Nb de pages à lire pour une tranversée d'index (ie. hauteur de l'arbre)
  - k : Ordre du l'arbre B
  - ϕ : *clustering factor* (cf. indexation)
  - *Sel* : Sélectivité d'un attribut
  - Δ : Nb de feuilles de $\mathcal{I}$ sélectionnées
- $|\mathcal{M}|$: Nb de pages tampons en MC pour la lecture

Plan

- Qu'est-ce qu'un index ?
- Index dense *versus* Index non dense
- Arbre-B+
- Index de hachage
- Index Bitmap

Principe d'indexation : Exemple de fichier

- Soit le fichier **Cours** contenant des données
- *Cours* (CODE, Intitule, Responsable, nb_auditeur)
- On souhaite accéder :
  - Au Responsable/Intitule d'un cours grâce à son CODE
  - Aux intitulés des cours d'un responsable
  - Aux nombre de cours ayant un certain nombre d'auditeurs

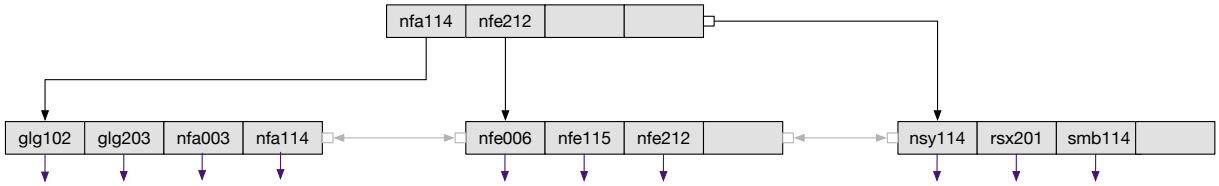
⇒ Besoins d'accès particuliers

Principe d'indexation : Exemple du fichier *Cours*

Fichier stocké sur 3 pages disques.

Page	CODE	Intitule	Responsable	Élèves
P1	NFE106	Ingénierie et optimisation de Base de Données	Nicolas Travers	30
	NFP107	Initiation à la Base de Données	Michel Crucianu	100
	NFE204	Bases de données documentaires et distribuées	Philippe Rigaux	50
P2	NFE205	Base de Données Avancées	Michel Crucianu	20
	NFE156	Pratiques et Outils de DBA	Nicolas Travers	15
	NFA011	Dev. d'appli. pour les bases de données	Marin Ferecatu	40
P3	NFE102	Infra. Techno. pour le Commerce Electronique	Philippe Rigaux	20
	NSY135	Patrons, Framework, ORM	Philippe Rigaux	30
	NFE209	SI - Audit et Gouvernance	Elisabeth Métais	25

Exemple d'index



Principe d'indexation : Index

- Un index est stocké dans un autre fichier $\mathcal{I}$
- Besoin d'une clé d'accès $\mathcal{K}$: *CODE*
- Besoin d'un pointeur sur les données (ROWID) : `@`
- Caractéristiques :
  - $\mathcal{I}$ ne contient pas les données
  - A chaque $\mathcal{K}$ correspond un ou plusieurs ROWIDS `@`
  - $\mathcal{I}$ est ordonné sur les valeurs de $\mathcal{K}$
  - Entrée de $\mathcal{I}$: ($\mathcal{K}$, `@`)
  - Si $\mathcal{I}$ plus gros qu'une page, on indexe de la même manière chaque page de $\mathcal{I}$
- `ROWID` :
  - Pointeur logique sur une instance
  - N°page + N° tuple + N° objet + N° fichier

Accès aux données

Deux étapes pour faire une recherche :

1. `INDEX RANGE SCAN` : Parcours de l'index
   - Donne l'adresse d'une page $\mathcal{P}$
   - On traverse l'arbre jusqu'aux feuilles de $\mathcal{I}$
   - Complexité : hauteur de l'arbre (Nb de pages)
   - Retour : liste de ROWIDS correspondant
2. `TABLE ACCESS BY ROWIDS` : Accès direct aux données
   - Ouverture de la page `@`
   - Parcours séquentiel de la page pour retrouver la(les) valeur(s) de $\mathcal{K}$

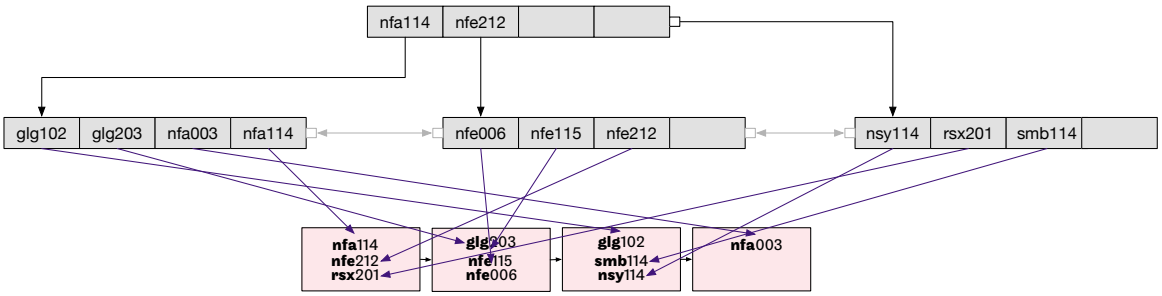
Index dense *versus* Index non dense

- **Index non dense**[6] : `Une page = un pointeur`
  - Valeurs ordonnées physiquement sur la clé $\mathcal{K}$
  - Une clé de l'index = premier tuple de chaque page
  - Très peu d'accès
  - Peu de place : $||\mathcal{K}|| = |\mathcal{R}|$
  - Un seul ordonnancement = un seul index non dense
- **Index dense** : `Une clé = un pointeur`
  - ROWID des feuilles très hétérogènes
  - Volumineux : $||\mathcal{K}|| = ||\mathcal{R}||$
  - Temps de parcours lent pour des valeurs multivaluées

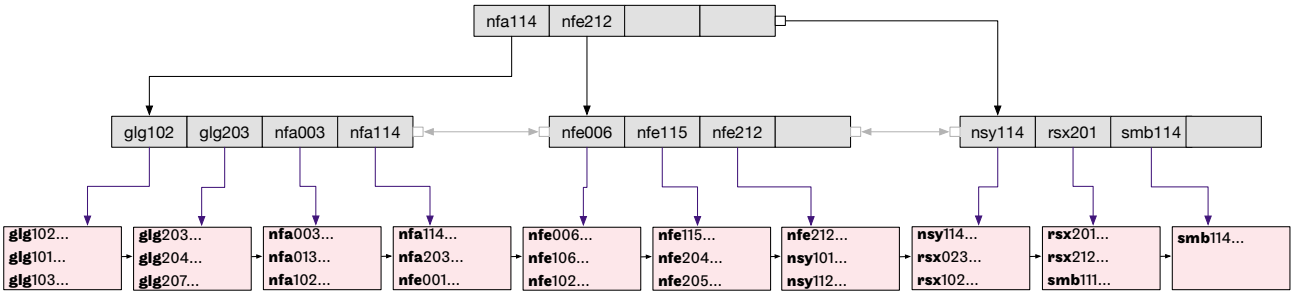
$\Rightarrow$ Quel est le type de l'index précédent ?

6. *Organization Index/IOT* (Oracle), *Clustered index* (SQLServer/DB2), *InnoDB* (MySQL)...

Index dense *versus* Index non dense



BTree dense : une clé = un pointeur



BTree non-dense : une page = un pointeur

Arbre B+

- *Balanced Tree* : Arbre équilibré
- Implanté dans tous les SGBD relationnels (index par défaut)
- Les feuilles de l'arbre contiennent les pointeurs
- Principe des Arbres Binaires de Recherche

⇒ **Optimise les requêtes d'égalité**
(avec peu de valeurs, et les inéquations)

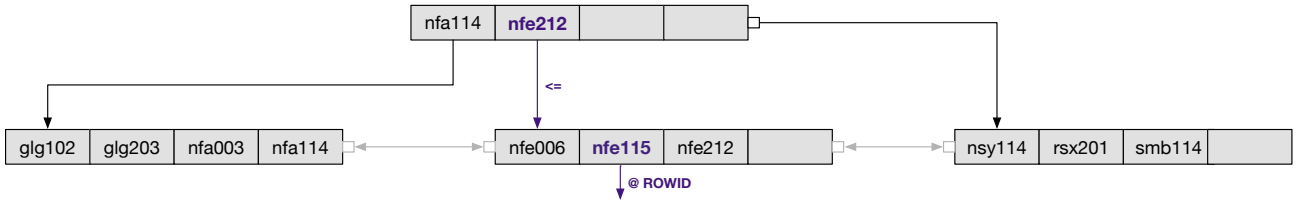
Arbre B+ : caractéristiques

- **Noeud** : clés + ROWID
- **Noeuds intermédiaires** : ROWID sur d'autres noeuds
- **Noeuds feuilles** : ROWID sur tuples du fichier indexé
- **Taille d'un noeud** : 2 * k valeurs (k est l'**ordre** de l'arbre)
- Quel est l'ordre de l'arbre de l'exemple ?

Arbre B+ : Recherche

- Recherche récursive de $\mathcal{K}$ à partir de la racine
- Soit C_1 à C_n les valeurs des clés de la page
  1. Si $\kappa \leq C_1$, recherche sur le noeud référencé P_1
  2. Si $\kappa > C_n$, recherche sur le noeud référencé P_{n+1}
  3. Si $C_i < \kappa \leq C_{i+1}$, recherche sur le noeud référencé P_{i+1}

Arbre B+ : Recherche de *nfe115*



Arbre B+ : Insertion

- Insertion de $(\kappa, @)$, dans Arbre B+ d'ordre k :
  1. **Recherche** de la feuille correspondant à $\mathcal{K}$
  2. **Insertion** ordonnée.
     Si Nb de clés de la page $p = 2 * k + 1$:
     1. Allocation d'une nouvelle page p'
     2. Les $k + 1$ premiers articles sont mis dans p
     3. Les k derniers articles sont mis dans p'
     4. La clé $k + 1$ est insérée au père de p.
        Pointeur gauche sur p, pointeur droit sur p'.
     5. Si père de p déborde, recommencer l'étape **insertion** sur *père de p*.

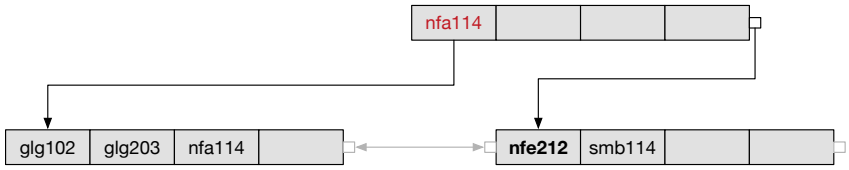
Arbre B+ : Exemple de création

glg102, glg203, nfa114, smb114, nfe212, nfe006, nfa003, nfe115, nsy114, rsx201

Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

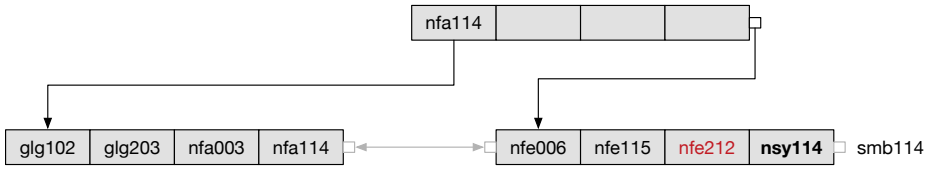
Arbre B+ : Exemple de création

glg102, glg203, nfa114, smb114, nfe212, nfe006, nfa003, nfe115, nsy114, rsx201



Arbre B+ : Exemple de création

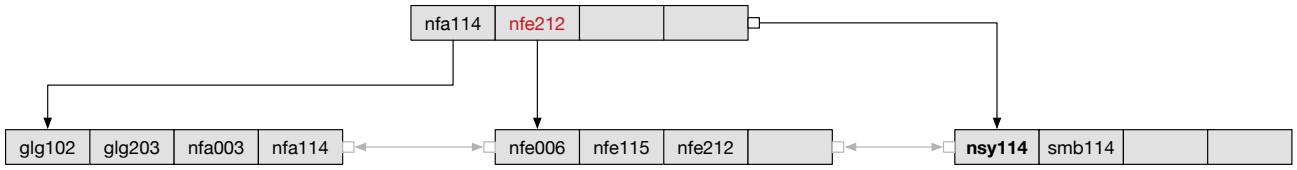
glg102, glg203, nfa114, smb114, nfe212, nfe006, nfa003, nfe115, nsy114, rsx201



Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

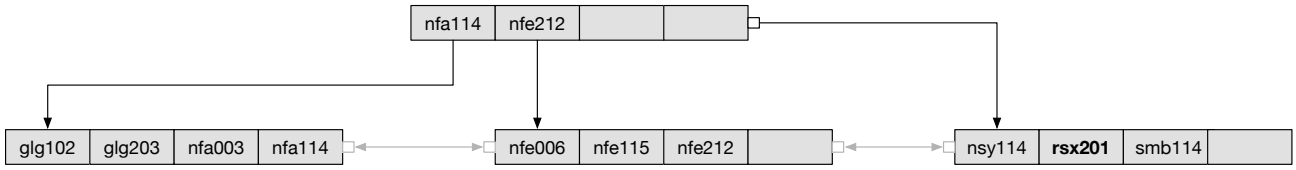
Arbre B+ : Exemple de création

glg102, glg203, nfa114, smb114, nfe212, nfe006, nfa003, nfe115, nsy114, rsx201



Arbre B+ : Exemple de création

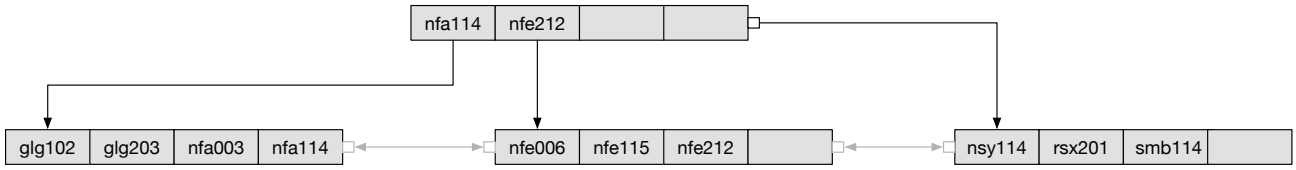
glg102, glg203, nfa114, smb114, nfe212, nfe006, nfa003, nfe115, nsy114, rsx201



Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

Arbre B+ : Exemple de création

glg102, glg203, nfa114, smb114, nfe212, nfe006, nfa003, nfe115, nsy114, rsx201



Exercice :insérer NFA025, NFE214, NSY012, NFE207, NFE102

Arbre B+ : Ordre et Hauteur

- L'ordre k : dépend de la taille du noeud (nb de clés)
  - $\mathcal{K}$: taille de la clé à indexer
  - Case : entete + (entete d'attribut + $\mathcal{K}$) + ROWID [7]
  - Nb de cases par noeud : $\left\lfloor \frac{|DataBlock|}{3+(x+|cle|)+10} \right\rfloor$
  - Ordre k : $\frac{NbCases}{2}$
- Hauteur de l'index :
  - Dépend du nombre de clés : $||\mathcal{K}||$
  - Division de l'ensemble par k à chaque niveau
  - Division récursive = logarithme
  - $\mathcal{I} = \lceil \log_{ordre}(||\mathcal{K}||) \rceil$
  - Index dense [8] : $||\mathcal{K}|| \leq ||\mathcal{R}||$
  - Index non-dense : $||\mathcal{K}|| = |\mathcal{R}|$

7. ROWID = 10 octets sous Oracle
8. Les valeurs nulles ne sont pas indexées

Creation d'un BTree

- Btree dense :
  CREATE INDEX *Cours_nom* ON *Cours* (*nom*) ;
- Btree non-dense sous `Oracle` :
  CREATE TABLE Cours (...) *ORGANIZATION INDEX* ;

Hachage

- Partitionnement des données
- Très peu de place en mémoire
  - Fonction de hachage
  - Table d'adresses de partitions
- Collisions : plusieurs clés $\mathcal{K}$ par partitions
- Divise le temps de parcours par le nombre de partitions (en moyenne)

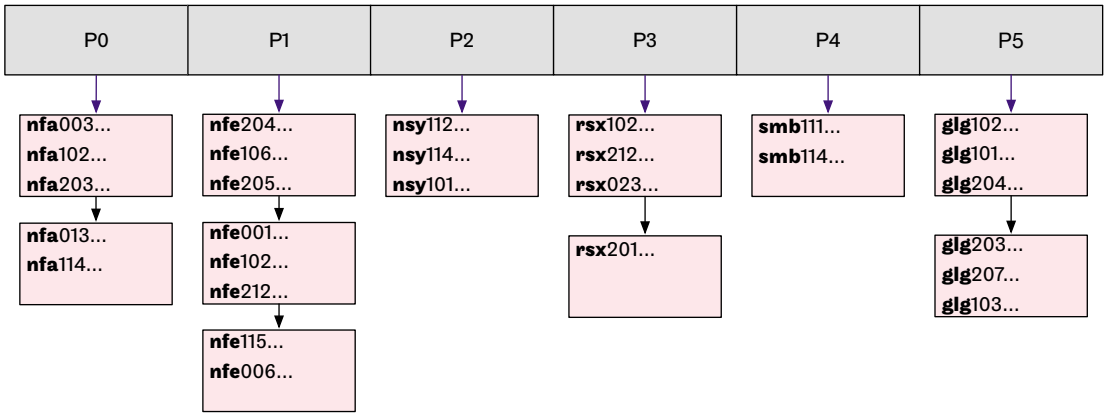
$\Rightarrow$ **Optimise les requêtes d'égalité**.
(Avec de nombreuses valeurs identiques)

Hachage : Caractéristiques

- **Fonction de hachage** : $h(\mathcal{K}) = \alpha$
  - $1 \leq \alpha \leq \beta$
  - β = Nb de partitions
  - h est une fonction uniforme[9] : $P(h(\kappa) = \alpha) = \frac{1}{\beta}$
  - Si $h(\mathcal{K}) = \alpha$, alors on ajoute la clé $\mathcal{K}$ (tout le n-uplet) à la page α
- **Table de hachage** T en MC[10]
  - $T[\alpha]$ = Adresse du bloc α
- Taille d'une partition : $\left\lceil \frac{|\mathcal{R}|}{\beta} \right\rceil$

9. Par défaut, *ORA_HASH* sous `Oracle`
`http://docs.oracle.com/cd/B28359_01/server.111/b28286/functions112.htm`
10. Mémoire Centrale

Hachage : Exemple



Partitionnement

- Plusieurs types de partitionnement :
  - Hachage
  - Intervalles de valeurs
  - Liste de valeurs
  - Composition des précédentes
- Exemple de création :
  CREATE TABLE Cours (...)
  PARTITION BY HASH (RESPONSABLE) PARTITIONS 20 ;

Bitmap

- Permet de compter le nombre de tuples ayant une certaine valeur
- Très peu de place en mémoire
- Evite de parcourir les données

⇒ **Optimise les requêtes d'agrégats**.
(Ayant une cardinalité faible)

Bitmap : Motivation

- On cherche à optimiser les requêtes :
  - SELECT count(*) FROM Cours WHERE responsable = 'Nicolas Travers' ;
  - Faible cardinalité de la valeur recherchée (comparé au nombre de n-uplets)
- **Arbre B+** :
  - Beaucoup de n-uplets ayant même valeur
  - Feuilles de l'arbre très grande
  - Risque d'accès aux données de chaque pointeur !
- **Table de hachage** :
  - Beaucoup de collisions dans les partitions

Bitmap : Structure

- **Bitmap** sur l'attribut *responsable*
  - $||\mathcal{R}||$: Nombre de tuples
  - $card_{resp}$: Cardinalité de l'attribut Responsable (nb valeur distinctes)
- Création de $card_{resp}$ **vecteurs bitmap** de $||\mathcal{R}||$ bits
  - Si le i^{eme} tuple a pour valeur 'Nicolas Travers', alors le i^{eme} bit du vecteur 'Nicolas Travers' a pour valeur 1
  - Le i^{eme} bit des autres vecteurs a pour valeur 0

Bitmap : Exemple

Nicolas Travers	Michel Crucianu	Philippe Rigaux	Marin Ferecatu	Elisabeth Métais
1	0	0	0	0
0	**1**	0	0	0
0	0	**1**	0	0
0	**1**	0	0	0
1	0	0	0	0
0	0	0	**1**	0
0	0	**1**	0	0
0	0	**1**	0	0
0	0	0	0	**1**

Index Bitmap sur *Responsable*

Bitmap : Caractéristiques

- **Stockage** :
  - Codage par tranche de 8 bits
  - $\Rightarrow$ Taille d'un vecteur en octets : $\left\lceil \frac{||\mathcal{R}||}{8} \right\rceil$
  - Page de taille $8ko$ (moins l'en-tête)
  - $\Rightarrow$ Taille d'un vecteur en pages : $\left\lceil \frac{||\mathcal{R}||}{8 \times 8000} \right\rceil$
  - Taille de l'index : $card_{resp} \times \left\lceil \frac{||\mathcal{R}||}{64000} \right\rceil$ pages
- Création d'un Bitmap :
  CREATE BITMAP INDEX Cours_Nom_bitmap ON Cours (NOM) ;

Bitmap : Requêtes d'agrégat

- **SELECT count(*) FROM Cours WHERE responsable = 'Nicolas Travers' ;**
  - Il suffit de compter le nombre de bits à **1** dans la première colonne
- **SELECT count(*) FROM Cours WHERE responsable = 'Nicolas Travers' OR responsable = 'Michel Crucianu' ;**
  - $\Rightarrow$ *OU* logique avec les deux vecteurs de *Responsable*
- **SELECT count(*) FROM Cours WHERE responsable = 'Nicolas Travers' AND NB_auditeur = 20 ;**
  - $\Rightarrow$ *ET* logique entre vecteur de *Responsable* et vecteur de *NB_auditeur*

Bitmap : Requêtes d'agrégat (2)

- **SELECT count(*) FROM Cours WHERE NB_auditeurs > 20 ;**
  - ⇒ *OU* logique entre les vecteurs *NB_auditeurs* ayant valeur > 20
- **SELECT count(*) FROM Cours WHERE responsable = 'Nicolas Travers' AND NB_auditeur > 20 ;**
  - ⇒ *ET* logique entre vecteur de *Responsable* et résultat du vecteur *NB_auditeur* > 20

Index : Récapitulatif

- **Requêtes très sélectives**
  - Arbre B+ dense (ie. clé primaire)
  - Arbre B+ non-dense (ie. beaucoup de données statiques ou incrémentales)
- **Requêtes sélectives**
  - Arbre B+ (ie. peu multivaluée)
  - Hachage (ie. beaucoup de valeurs)
- **Requêtes avec intervalle**
  - Arbre B+
- **Requêtes avec agrégats**
  - Bitmap

Index : Bonus

- **Index couvrant**
  - Ajoute une donnée associée à la clé
  - Permet d'éviter un accès aux données
  - Prend plus de place
- **Requêtes multi-critères**
  1. un Arbre B+ pour chaque attribut du prédicat (2x `INDEX RANGE SCAN`)
  2. Intersection des pointeurs retournés (`AND-EQUAL`)
  3. Accès aux pages du résultat (`TABLE ACCESS BY ROWIDS`)

Opérateurs Relationnels

- Une requête SQL est transformée en Plan d'exécution
- Le plan est composé d'opérateurs
- Chaque opérateur à un coût d'évaluation
  - Coût en Entrées/Sorties
  - Dépend des statistiques (taille des tables, nb tuples, cardinalité, sélectivité...)

Sélection : Algorithmes

- Opération : $\sigma_{a='x'}(\mathcal{R})$
- Sélection Séquentielle : `TABLE ACCESS FULL`
- Sélection avec Index Arbre B+ : `INDEX RANGE SCAN`
- Sélection par Hachage : `PARTITION HASH SINGLE`
- Impact de la sélectivité sur nombre de tuples en sortie et/ou le nombre d'accès

Sélectivité : *Sel*

- **Pourcentage** de n-uplets pour une **valeur** cherchée
- Dépend du prédicat de sélection et des données : $\sigma_{a='x'}(\mathcal{R})$
- Par défaut :
  - $Sel(\mathcal{R}\ (a))$ [11] $= \frac{1}{Card(\mathcal{R}(a))}$
  - Exemple :
    - $\mathcal{R}\ (a)$ est réparti sur 20 valeurs différentes (cardinalité = 20)
    - Statistiquement : $Sel(\mathcal{R}\ (a)) = \frac{1}{20} = 5\%$
  - Clé primaire : $Sel = \frac{1}{||\mathcal{R}||}$ (INDEX *UNIQUE* SCAN)
- Statistiques :
  - Histogramme de fréquences par valeur ($\mathcal{R}$(a))

NFA011	NFE102	NFE106	NFE204	NFE205	NFP107	NSY218
5%	2%	6%	4%	3%	2.5%	1.2%

⚠ Coût de calcul / mise à jour (requêtes d'agrégation)

11. $Card(\mathcal{R}(a))$: nb de valeurs distinctes

TABLE ACCESS FULL : Sélection Séquentiel

- Appliquer si aucun index sur $\mathcal{R}(a)$
- Pour chaque page b de $\mathcal{R}$
  - Pour chaque tuple t de b
    - Si $t.a =' x'$
      Ajouter t dans $\mathcal{S}$

$\Rightarrow$ Complexité ? $|\mathcal{R}|$

TABLE ACCESS FULL : $\sigma_{resp=' MichelCrucianu'}(Cours)$

Page	CODE	Intitule	Responsable
P1	NFE106	Ingénierie et optimisation de Base de Données	Nicolas Travers
	NFP107	Initiation à la Base de Données	Michel Crucianu
	NFE204	Bases de données documentaires et distribuées	Philippe Rigaux
P2	NFE205	Base de Données Avancées	Michel Crucianu
	NFE156	Pratiques et Outils de DBA	Nicolas Travers
	NFA011	Développement d'applications pour bases de données	Marin Ferecatu
P3	NFE102	Infrastructures Technologiques pour le commerce...	Philippe Rigaux
	NSY135	Patrons, Framework, ORM	Philippe Rigaux
	NFE209	SI - Audit et Gouvernance	Elisabeth Métais

Tampon de sortie

CODE	Intitule	Responsable
NFP107	Initiation à la Base de Données	Michel Crucianu

- Cardinalité de 'responsable' ?
- Sélectivité moyenne ?
- Sélectivité de $\sigma_{resp=' MichelCrucianu'}(Cours)$?

TABLE ACCESS FULL : $\sigma_{resp='MichelCrucianu'}(Cours)$

Page	CODE	Intitule	Responsable
P1	NFE106	Ingénierie et optimisation de Base de Données	Nicolas Travers
	NFP107	Initiation à la Base de Données	Michel Crucianu
	NFE204	Bases de données documentaires et distribuées	Philippe Rigaux
P2	NFE205	Base de Données Avancées	Michel Crucianu
	NFE156	Pratiques et Outils de DBA	Nicolas Travers
	NFA011	Développement d'applications pour bases de données	Marin Ferecatu
P3	NFE102	Infrastructures Technologiques pour le commerce...	Philippe Rigaux
	NSY135	Patrons, Framework, ORM	Philippe Rigaux
	NFE209	SI - Audit et Gouvernance	Elisabeth Métais

Tampon de sortie

CODE	Intitule	Responsable
NFP107	Initiation à la Base de Données	Michel Crucianu
NFE205	Base de Données Avancées	Michel Crucianu

- Cardinalité de 'responsable' ?
- Sélectivité moyenne ?
- Sélectivité de $\sigma_{resp='MichelCrucianu'}(Cours)$?

Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

TABLE ACCESS FULL : $\sigma_{resp='MichelCrucianu'}(Cours)$

Page	CODE	Intitule	Responsable
P1	NFE106	Ingénierie et optimisation de Base de Données	Nicolas Travers
	NFP107	Initiation à la Base de Données	Michel Crucianu
	NFE204	Bases de données documentaires et distribuées	Philippe Rigaux
P2	NFE205	Base de Données Avancées	Michel Crucianu
	NFE156	Pratiques et Outils de DBA	Nicolas Travers
	NFA011	Développement d'applications pour bases de données	Marin Ferecatu
P3	NFE102	Infrastructures Technologiques pour le commerce...	Philippe Rigaux
	NSY135	Patrons, Framework, ORM	Philippe Rigaux
	NFE209	SI - Audit et Gouvernance	Elisabeth Métais

Tampon de sortie

CODE	Intitule	Responsable
NFP107	Initiation à la Base de Données	Michel Crucianu
NFE205	Base de Données Avancées	Michel Crucianu

- Cardinalité de 'responsable' ?
- Sélectivité moyenne ?
- Sélectivité de $\sigma_{resp='MichelCrucianu'}(Cours)$?

67 / 284
le cnam
Nicolas Travers

TABLE ACCESS FULL : $\sigma_{resp='MichelCrucianu'}(Cours)$

Page	CODE	Intitule	Responsable
P1	NFE106	Ingénierie et optimisation de Base de Données	Nicolas Travers
	NFP107	Initiation à la Base de Données	Michel Crucianu
	NFE204	Bases de données documentaires et distribuées	Philippe Rigaux
P2	NFE205	Base de Données Avancées	Michel Crucianu
	NFE156	Pratiques et Outils de DBA	Nicolas Travers
	NFA011	Développement d'applications pour bases de données	Marin Ferecatu
P3	NFE102	Infrastructures Technologiques pour le commerce...	Philippe Rigaux
	NSY135	Patrons, Framework, ORM	Philippe Rigaux
	NFE209	SI - Audit et Gouvernance	Elisabeth Métais

Tampon de sortie

CODE	Intitule	Responsable
NFP107	Initiation à la Base de Données	Michel Crucianu
NFE205	Base de Données Avancées	Michel Crucianu

- Cardinalité de 'responsable' ? 6
- Sélectivité moyenne ? $\frac{1}{6} = 16,6\%$
- Sélectivité de $\sigma_{resp='MichelCrucianu'}(Cours)$? $\frac{2}{9} = 22,2\%$

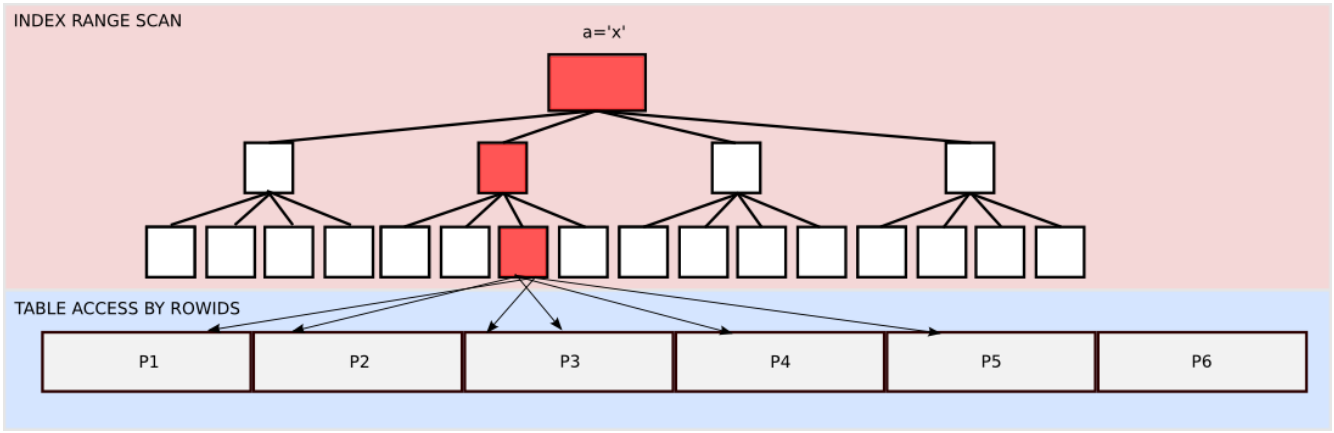
INDEX RANGE SCAN + TABLE ACCESS BY INDEX ROWID

- Sélection par index appliqué si $\mathcal{I}$ sur $\mathcal{R}(a)$
- `INDEX RANGE SCAN`
  - Ouvrir la page $\mathcal{P}$ racine de $\mathcal{I}$
    Tant que $\mathcal{P}$ n'est pas une feuille
    Parcourir l'arbre de $\mathcal{P}$ avec a='x'
    $\mathcal{P}$ = nouvelle page cible
  - Sortie : $\mathcal{L}$ = liste de ROWIDS
    - Selectivité de l'attribut a : Sel
    - Pourcentage de sélection : $0 < Sel < 1$

    $\Rightarrow$ $||\mathcal{L}|| = ||ROWID|| \times Sel$
- `TABLE ACCESS BY INDEX ROWID`
  - Pour chaque l de $\mathcal{L}$
    Ouvrir la page p pointée par l
    Pour chaque tuple t de p
    Si $t.a =$ 'x'
    Ajouter t dans le tampon de sortie

$\Rightarrow$ Complexité : $|\mathcal{I}| + ||ROWID|| \times Sel$

INDEX RANGE SCAN + TABLE ACCESS BY INDEX ROWID : Exemple index dense



INDEX RANGE SCAN

- Parcours de l'index
  1. **Ordre** du BTree
     - Dépend de la taille de la clé
     - Un noeud = une page (8ko + metadata + PCTFREE)
     - Contient $2 \times k$ cases
     - Une case : Clé + ROWID
       Taille[12] : $3 + |cle| + 10$
     - ordre : $k = \left\lfloor \frac{(8192-192)\times 90\%}{3+|cle|+10} \right\rfloor \div 2$
  2. **Hauteur** du BTree
     - Division de l'espace par l'ordre à chaque niveau

     $\Rightarrow$ $log_k(||ROWID||)$
     Index dense : $||ROWID|| = ||\mathcal{R}||$
     Index non-dense : $||ROWID|| = |\mathcal{R}|$
- Plusieurs ROWIDS en sortie ($\mathcal{L}$)
  - Si $||\mathcal{L}|| > k \Rightarrow$ plusieurs feuilles : Δ
  - $\Delta = \left\lfloor \frac{||ROWID|| \times Sel}{ordre} \right\rfloor$

12. entête de tuple (3o), $|cle|$: taille des attributs clés (1+...), ROWID (10o)

INDEX RANGE SCAN + TABLE ACCESS BY INDEX ROWID : Delta

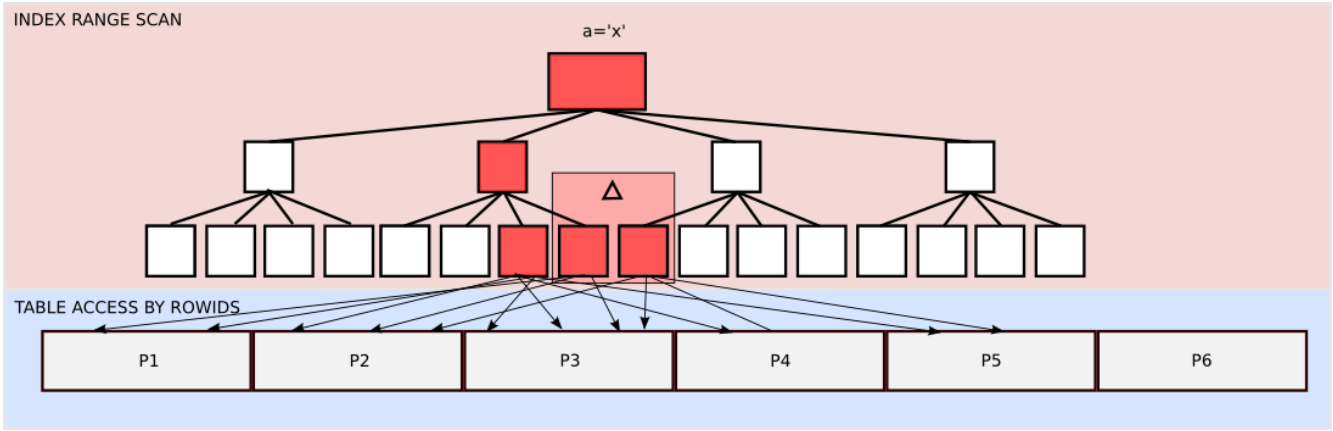


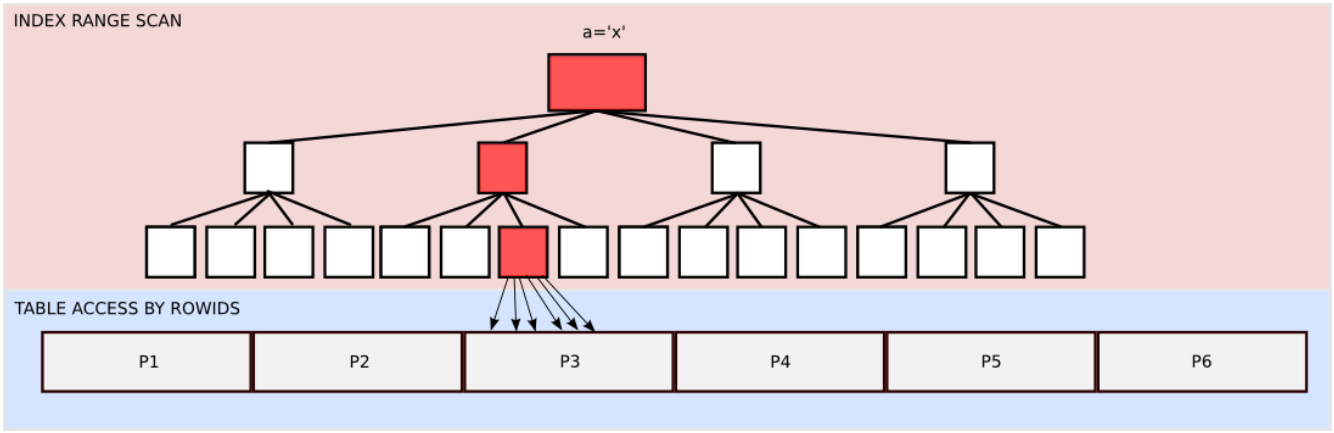
TABLE ACCESS BY INDEX ROWID

- Accès aux données via ROWIDS
- 1 ROWID = 1 page
- Optimisation :
  - Si 2 ROWIDS sur une même page = 1 accès
  - Tri des ROWIDS par page, puis accès
  - **clustering factor**[13] : ϕ
    - Degré de distribution aléatoire des données dans $\mathcal{R}$
    - $|\mathcal{R}| \leq \phi \leq ||\mathcal{R}||$
    - $\uparrow$ $\phi = |\mathcal{R}|$ (Index non-dense)
    - $\downarrow$ $\phi = ||\mathcal{R}||$ (Très aléatoire/dispersé) - valeur par défaut
    - Possibilité de trier les données physiquement

$\Rightarrow$ Nombre de pages accédées : $||\mathcal{R}|| \times Sel \times \frac{\phi}{||\mathcal{R}||} = Sel \times \phi$

13. facteur de foisonnement / regroupement des ROWIDS par page

INDEX RANGE SCAN + TABLE ACCESS BY INDEX ROWID : Clustering Factor



INDEX RANGE SCAN + TABLE ACCESS BY INDEX ROWID : Complexité

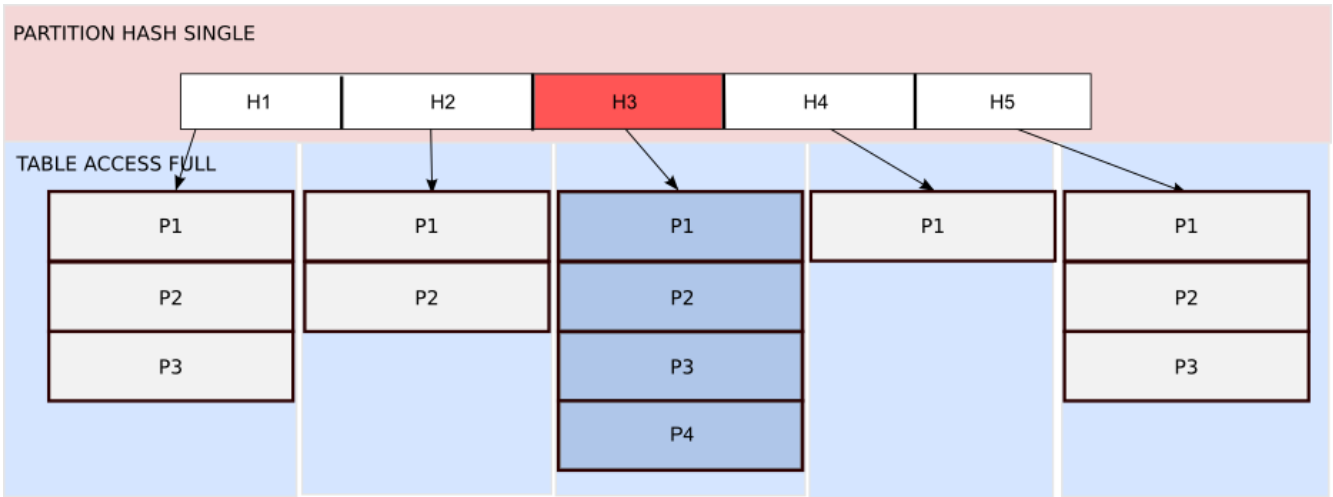
- Parcours de l'index : $|\mathcal{I}|$ (hauteur : $\log_k(||ROWID||)$)
- Nombre de feuilles : $\Delta = \left\lfloor \frac{||ROWID|| \times Sel}{k} \right\rfloor$
- Accès aux données : $\phi \times Sel$
- Coût moyen : $|\mathcal{I}| + \Delta + \phi \times Sel$
- Coût index unique : $Sel = \frac{1}{||\mathcal{R}||}$, $\Delta = 0$
- Coût index non dense : $\phi = |\mathcal{R}|$
- L'accès aux données ($\phi \times Sel$) n'est pas toujours obligatoire : regarder si les données requises sont dans l'index

PARTITION HASH SINGLE : Sélection par Hachage

- Appliquer si index Hash $\mathcal{H}$ sur $\mathcal{R}(a)$
- $\mathcal{P}$ partitions
- Données placées sur les partitions en fonction de $\mathcal{H}$
- Hacher la clé $'x'$: $\mathcal{H}('x') = v$ $(1 \leq v \leq \mathcal{P})$
- Recherche :
  Pour chaque page p de la partition $\mathcal{R}(v)$
  - Pour chaque tuple t de p
    - Si $t.a =' x'$
      Ajouter t dans $\mathcal{S}$

$\Rightarrow$ Complexité ? Taille d'une partition $\approx \left\lceil \frac{|\mathcal{R}|}{\mathcal{P}} \right\rceil$

PARTITION HASH SINGLE : Exemple



PARTITION HASH SINGLE : Partitionnement

- Répartition des clés en fonction de la fonction de hachage [14]
- N-uplets uniforméments répartis sur les $\mathcal{P}$ partitions

⚠ Problème de répartition

- Fonction inadaptée
- Répartition inégale des redondances [15]

⇒ Combinaison de hachage

⇒ Partitionnement par intervalles

14. Sous Oracle : `ORA_HASH`. Peut être surchargée
15. reste meilleur qu'un Arbre B+

SORT : Tri physique

Nécessaire pour :

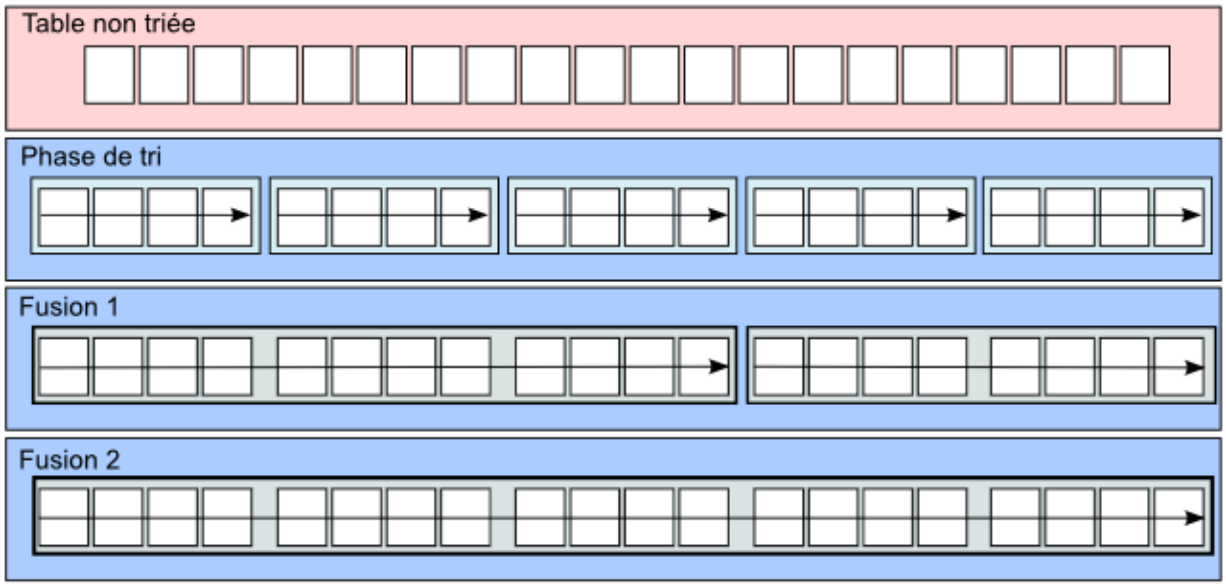
- Ordonner le résultat (Order By)
- Agrégation (Group By / Count,Sum,Avg ...)
- Éliminer les doublons (Distinct)
- Jointure par fusion (Sort Merge Join)

SORT : Deux phases

- $|\mathcal{M}|$: Tampons en mémoire
- $|\mathcal{R}'|$: Nombre de pages de $\mathcal{R}$ en entrée du tri [16]
- Si $|\mathcal{M}| < |\mathcal{R}'|$
  1. **Tri** :
     - On découpe $\mathcal{R}'$ en blocs de taille $|\mathcal{M}|$
     - On trie chaque bloc en mémoire
     - On écrit les blocs triés
     - On a $\frac{|\mathcal{R}'|}{|\mathcal{M}|}$ partitions triées
  2. **Fusion** :
     - On fusionne les partitions, $|\mathcal{M}| - 1$ blocs triés en même temps
     - Récursivement, on obtient au final une seule partition
     - Cette phase est la plus chère

16. Dépend de l'accès aux données (Full, Index, Hachage)
Calcul de pages normales projetées avec PCTFREE de 0%

SORT : Exemple avec $|\mathcal{M}| = 4$



SORT : Fusion de deux blocs

- Deux blocs L_1 et L_2 en mémoire, un tampon de sortie $\mathcal{S}$
- L_1 : 1, 5, 7, 9, 13, 27, 58, 89, 100, 101, 112+ pointeur p_1
- L_2 : 1, 2, 3, 4, 5, 10, 13, 20, 30, 31, 32, 57, 58+ pointeur p_2
- Tant que $p_1 \neq \emptyset \mid p_2 \neq \emptyset$
  - Si $\mathcal{S}$ plein $\Rightarrow$ écrire $\mathcal{S}$ sur le disque
  - Si ($p_1 \neq \emptyset$ & ($p_2 == \emptyset \mid val(p_1) <= val(p_2)$))
    Insérer $val(p_1)$ dans $\mathcal{S}$
    p_1 = suivant dans L_1
  - Sinon
    Insérer $val(p_2)$ dans $\mathcal{S}$
    p_2 = suivant dans L_2

SORT : Fusion de deux listes (2)

- Listes stockées sur plusieurs pages
- Algorithme identique, mais lorsque la page est vide, on passe à la suivante.
- Fusion de listes sur différentes pages

L_1	1, 4, 8	10, 13, 27	29, 55, 60	71, 85	
L_2	1, 2, 3	4, 5, 10	13, 28, 30	31, 32, 57	80

- Coût de lecture ? $|L_1| + |L_2|$
- Coût en ecriture ? $|L_1| + |L_2|$
- Besoin en MC ? 1 tampon d'écriture + $|\mathcal{M}| - 1$ en lecture

SORT : Phase de fusion

- **A chaque étape**
  - Fusion de $|\mathcal{M}| - 1$ partitions
    - Les $|\mathcal{M}| - 1$ listes sont fusionnées (et non deux)
    - Chaque liste fait $|\mathcal{M}|$ pages à la première étape
  - Résultat stocké dans le tampon
  - Résultat $|\mathcal{M}| - 1$ plus grand que l'étape précédente
  - $|\mathcal{M}| - 1$ partitions de moins
  - Lecture **et** écriture de $|\mathcal{R}'|$ pages
- Nombre d'étapes de fusion : $\left\lceil \log_{|\mathcal{M}|-1}(|\mathcal{R}'|) \right\rceil$
- Coût de la fusion : $2 \times |\mathcal{R}'| \times \left\lceil \log_{|\mathcal{M}|-1}(|\mathcal{R}'|) \right\rceil$

SORT : Exercice de tri fusion

18 pages, contenant chacune 3 valeurs. Trier avec $|\mathcal{M}| = 4$

30	28	5	3	2	5	6	30	25	4	2	3	9	2	20	15	21	12
19	32	15	16	20	16	10	40	21	30	39	10	3	8	25	12	42	6
7	18	25	21	12	7	1	42	27	21	20	17	8	9	6	8	39	8

Tri par blocs :

3	15	19	28	1	6	12	30	2	10	21	27	2	8	9	15	6	21
5	16	21	30	2	7	16	40	3	17	21	30	3	8	9	20	8	39
7	18	25	32	5	10	20	42	4	20	25	39	6	8	12	25	12	42

Fusion 1 :

1	3	5	7	10	16	18	20	21	27	30	39	2	6	8	9	15	21
2	3	5	7	12	16	19	21	25	28	30	40	3	8	8	12	20	39
2	4	6	10	15	17	20	21	25	30	32	42	6	8	9	12	20	42

Fusion 2 :

1	2	3	5	6	8	8	10	12	15	17	20	20	21	25	30	32	40
2	3	4	6	7	8	9	10	12	16	18	20	21	21	27	30	39	42
2	3	5	6	7	8	9	12	15	16	19	20	21	25	28	30	39	42

Tri : Coût total du tri fusion

- Accès à $\mathcal{R}$ (Full/Index/Hachage) : $\Rightarrow \mathcal{R}'$ en sortie
- Mémoire Centrale : $|\mathcal{M}|$
- Tri de $\frac{|\mathcal{R}'|}{|\mathcal{M}|}$ blocs de taille $|\mathcal{M}|$: $|\mathcal{R}'|$
- Fusion par $|\mathcal{M}| - 1$ blocs

  Nb de lectures : $|\mathcal{R}'| \times \left\lceil \log_{|\mathcal{M}|-1}(|\mathcal{R}'|) \right\rceil$

  Nb d'écritures : $|\mathcal{R}'| \times \left\lceil \log_{|\mathcal{M}|-1}(|\mathcal{R}'|) \right\rceil$
- Coût total : $(\text{Accès à } \mathcal{R} + |\mathcal{R}'|) + (2 \times |\mathcal{R}'| \times \left\lceil \log_{|\mathcal{M}|-1}(|\mathcal{R}'|) \right\rceil)$
- Si $|\mathcal{R}'| < |\mathcal{M}| \Rightarrow$ pas de fusion $=$ Accès à $\mathcal{R} + |\mathcal{R}'|$

Projection

Intégré dans plusieurs opérateurs :

- Sélection séquentielle : TABLE ACCESS FULL
- Sélection avec index : TABLE ACCESS BY ROWIDS
- Sélection par tri : SORT
- Jointures ne tenant pas en mémoire : NESTED LOOPS
- Jointures par hachage : HASH JOIN

Projection : Avantages

- Place mémoire
  - Taille du tuple minimum

  ⇒ Plus de tuples par page

  ⇒ Moins de pages temporaires à écrire et à lire
- Peu couteux en traitement
  - Pipeline [17]

⇒ Intégré dans tous les opérateurs lors d'une écriture en mémoire/disque

17. Section Plans/Pipeline

Projection : Exemple

- Table *Cours* :
  - 5000 tuples de taille 120 o $(3 + 3 \times (4 + 1) + 2 \times (50 + 1))$
  - PCTFREE de 10%
  - $|Cours| = \left\lceil \frac{5000}{\left\lfloor \frac{(8192-192)\times 90\%}{120} \right\rfloor} \right\rceil = 64$ pages
- Projection sur la Date (4 o)
  - Taille des tuples : 8 o $(3 + 1 \times (1 + 4))$
  - Ecriture temporaire : PCTFREE=0%
  - $|Cours'| = \left\lceil \frac{5000}{\left\lfloor \frac{(8192-192)}{8} \right\rfloor} \right\rceil = 5$ pages

HASH AGGREGATE

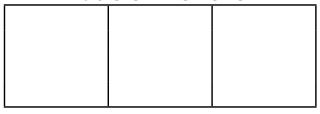
- Projection, hachage et incrément
  `SELECT Lettre, COUNT(*) FROM TABLEX GROUP BY Lettre`
- $|\mathcal{M}|$ Tampons en mémoire centrale
- $\mathcal{H}$ la fonction de hachage
- a l'attribut à projeter, grouper et compter
- Algo en 2 phases :
  1. Sélection/Projection
     Placement des résultats en mémoire dans un des $|\mathcal{M}| - 1$ tampons (partitions)
     Tampon choisi par $\mathcal{H}(a)$. Une fois plein, on écrit.
  2. Lecture de chaque partition générée

HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

- Compter l'occurence de chaque lettre
  - Projeter sur la lettre
  - Trouver la partition
  - Incrémenter son occurrence
- Capacité après projection : 1 page $\Rightarrow$ 4 lettres+incrément
- Fonction de hachage : indice de la lettre % 3
  - $h(l) = 0 \rightarrow$ c, f, i, l, o, r, u, x
  - $h(l) = 1 \rightarrow$ a, d, g, j, m, p, s, v, y
  - $h(l) = 2 \rightarrow$ b, e, h, k, n, q, t, w, z
- Une fois les partitions créées, parcours séquentiel avec somme des "count" locaux

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :



HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

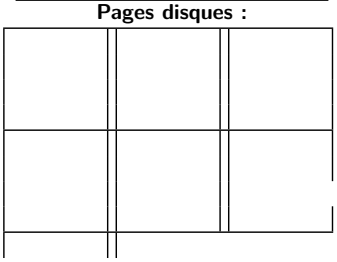
- Fonction de hachage : indice de la lettre % 3
  - $h(l) = 0 \rightarrow$ c, f, i, l, o, r, u, x
  - $h(l) = 1 \rightarrow$ a, d, g, j, m, p, s, v, y
  - $h(l) = 2 \rightarrow$ b, e, h, k, n, q, t, w, z

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 **K** 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

L 2 C 2 X 2 F 1	A 2 G 1 D 1 Y 1	B 2 H 2 E 2 Z 1

Pages disques :



Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

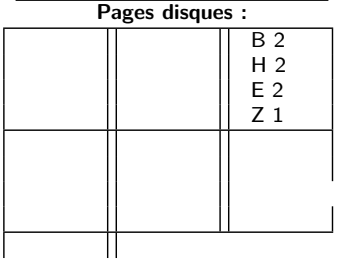
- Fonction de hachage : indice de la lettre % 3
  - $h(l) = 0 \rightarrow$ c, f, i, l, o, r, u, x
  - $h(l) = 1 \rightarrow$ a, d, g, j, m, p, s, v, y
  - $h(l) = 2 \rightarrow$ b, e, h, k, n, q, t, w, z

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 **P** 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

L 2 C 3 X 2 F 1	A 2 G 2 D 1 Y 1	K 1 B 1

Pages disques :



HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

- Fonction de hachage : indice de la lettre % 3
  - $h(l) = 0 \rightarrow$ c, f, i, l, o, r, u, x
  - $h(l) = 1 \rightarrow$ a, d, g, j, m, p, s, v, y
  - $h(l) = 2 \rightarrow$ b, e, h, k, n, q, t, w, z

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 **U** 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

L 2 C 3 X 2 F 1	P 1	K 1 B 1

Pages disques :

	A 2 G 2 D 1 Y 1	B 2 H 2 E 2 Z 1

HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

- Fonction de hachage : indice de la lettre % 3
  - $h(l) = 0 \rightarrow$ c, f, i, l, o, r, u, x
  - $h(l) = 1 \rightarrow$ a, d, g, j, m, p, s, v, y
  - $h(l) = 2 \rightarrow$ b, e, h, k, n, q, t, w, z

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 **H** 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

U 2	P 1	K 1 B 1 N 1 Z 1

Pages disques :

L 2 C 3 X 2 F 1	A 2 G 2 D 1 Y 1	B 2 H 2 E 2 Z 1

HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

- Fonction de hachage : indice de la lettre % 3
  - $h(l) = 0 \rightarrow$ c, f, i, l, o, r, u, x
  - $h(l) = 1 \rightarrow$ a, d, g, j, m, p, s, v, y
  - $h(l) = 2 \rightarrow$ b, e, h, k, n, q, t, w, z

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 **Z** 1

Table en mémoire :

U 2 I 1 L 1 X 1	P 1 A 2 S 1	H 1 Z 1

Pages disques :

L 2 C 3 X 2 F 1	A 2 G 2 D 1 Y 1	B 2 H 2 E 2 Z 1
	N 1	K 1 B 1 Z 1

Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

Pages disques :

L 2 C 3 X 2 F 1	A 2 G 2 D 1 Y 1	B 2 H 2 E 2 Z 1
U 2 I 1 L 1 X 1	P 1 A 2 S 1 N 1	K 1 B 1 Z 1
		H 1 Z 1

92 / 284
le cnam
Nicolas Travers

HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

Pages disques :

L 2 C 3 X 2 F 1	A 2 G 2 D 1 Y 1	B 2 H 2 E 2 Z 1
U 2 I 1 L 1 X 1	P 1 A 2 S 1 N 1	K 1 B 1 Z 1
		H 1 Z 1

Pages mémoire :

L 3 C 3 X 3 F 1	U 2 I 1	

Résultat :

HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

Pages disques :

	A 2 G 2 D 1 Y 1	B 2 H 2 E 2 Z 1
	P 1 A 2 S 1 N 1	K 1 B 1 Z 1
		H 1 Z 1

Pages mémoire :

A 4 G 2 D 1 Y 1	P 1 S 1	

Résultat :

L 3 C 3 X 3 F 1	U 2 I 1			

HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

Pages disques :

		B 2 H 2 E 2 Z 1
	N 1	K 1 B 1 Z 1
		H 1 Z 1

Pages mémoire :

B 3 H 3 E 2 Z 3	K 1 N 1	

Résultat :

L 3 C 3 X 3 F 1	U 2 I 1 A 4 G 2	D 1 Y 1 P 1 S 1		

Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

HASH AGGREGATE : Exercice $|\mathcal{M}| = 4$

Pages disques :

Pages mémoire :

Résultat :

L 3 C 3 X 3 F 1	U 2 I 1 A 4 G 2	D 1 Y 1 P 1 S 1	B 3 H 3 E 2 Z 3	K 1 N 1

HASH AGGREGATE : Complexité

- Hachage sur résultat (partitions à créer)
- Accès aux données de $\mathcal{R}$: $\Rightarrow |\mathcal{R}'|$
- Nombre de partition : $|\mathcal{M}| - 1$
- Taille d'une partition : $\left\lceil \frac{|\mathcal{R}'|}{|\mathcal{M}|-1} \right\rceil$
- Projection/Partitionnement : $(|\mathcal{M}| - 1) \times \frac{|\mathcal{R}'|}{|\mathcal{M}|-1} \simeq |\mathcal{R}'|$
- Tri des partitions : $(|\mathcal{M}| - 1) \times \frac{|\mathcal{R}'|}{|\mathcal{M}|-1} \simeq |\mathcal{R}'|$
- Coût total : Accès à $\mathcal{R} + 2 \times |\mathcal{R}'|$
- Problème : $\frac{|\mathcal{R}'|}{|\mathcal{M}|-1} > |\mathcal{M}|$, tri en mémoire impossible

HASH AGGREGATE : Amélioration

- Si $\frac{|\mathcal{R}'|}{|\mathcal{M}|-1} > |\mathcal{M}|$
  - Hachage récursif sur chaque partition
  - Fonction h' différente
  - Algorithme identique
- Coût final : Accès à $\mathcal{R} + 2 \times |\mathcal{R}'| + 2 \times \alpha \times |\mathcal{R}'|$

  $\alpha \geq 0$ (nombre de hachages successifs)

SORT UNIQUE : Tri de données avec DISTINCT

- Projection, Tri et suppression des doublons
  `SELECT DISTINCT Lettre FROM TABLEX GROUP BY Lettre`
- $|\mathcal{M}|$ Tampons en mémoire centrale
- Algo en 2 étapes :
  1. Lecture page par page de $\mathcal{R}$
     - **Projection** des valeurs dans $|\mathcal{M}| - 1$ pages
     - **Élimination de doublons et tri** sur $|\mathcal{M}| - 1$
     - Quand $|\mathcal{M}| - 1$ plein, écrire sur le disque (une liste pour la fusion)
  2. Tri fusion des blocs projetés
     - **Élimination de doublons lors du parcours**

SORT UNIQUE : Exercice $|\mathcal{M}| = 3$

- On souhaite projeter et éliminer les doublons de la seconde colonne de la table (*lettre*)
- Capacité après projection : 1 page $\Rightarrow$ 4 lettres

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 **G** 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

A B C E	F H L X

SORT UNIQUE : Exercice $|\mathcal{M}| = 3$

- On souhaite projeter et éliminer les doublons de la seconde colonne de la table (*lettre*)
- Capacité après projection : 1 page $\Rightarrow$ 4 lettres

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 **K** 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

B D G H	L X Y Z

Stockage temporaire
1) Phase de Tri

A B C E	F H L X

SORT UNIQUE : Exercice $|\mathcal{M}| = 3$

- On souhaite projeter et éliminer les doublons de la seconde colonne de la table (*lettre*)
- Capacité après projection : 1 page $\Rightarrow$ 4 lettres

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 **H** 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

B C G K	N P U Z

Stockage temporaire
1) Phase de Tri

A B C E	F H L X	B D G H	L X Y Z

SORT UNIQUE : Exercice $|\mathcal{M}| = 3$

- On souhaite projeter et éliminer les doublons de la seconde colonne de la table (*lettre*)
- Capacité après projection : 1 page $\Rightarrow$ 4 lettres

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 **Z** 1

Table en mémoire :

A H I L	S X Z

Stockage temporaire
1) Phase de Tri

A B C E	F H L X	B D G H	L X Y Z	B C G K	N P U Z

SORT UNIQUE : Exercice $|\mathcal{M}| = 3$

- On souhaite projeter et éliminer les doublons de la seconde colonne de la table (*lettre*)
- Capacité après projection : 1 page $\Rightarrow$ 4 lettres

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

Stockage temporaire
1) Phase de Tri

A B C E	F H L X	B D G H	L X Y Z	B C G K	N P U Z	A H I L	S X Z

SORT UNIQUE : Exercice $|\mathcal{M}| = 3$

- On souhaite projeter et éliminer les doublons de la seconde colonne de la table (*lettre*)
- Capacité après projection : 1 page $\Rightarrow$ 4 lettres

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

Stockage temporaire
1) Phase de Tri

A B C E	**F H L X**	**B D G H**	**L X Y Z**	B C G K	N P U Z	A H I L	S X Z

2) Phase de Fusion

A B C D	E F G H	L X Y Z

Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

SORT UNIQUE : Exercice $|\mathcal{M}| = 3$

- On souhaite projeter et éliminer les doublons de la seconde colonne de la table (*lettre*)
- Capacité après projection : 1 page $\Rightarrow$ 4 lettres

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

Stockage temporaire
1) Phase de Tri

A B C E	F H L X	B D G H	L X Y Z	**B C G K**	**N P U Z**	**A H I L**	**S X Z**

2) Phase de Fusion

A B C D	E F G H	L X Y Z	A B C G	H I K L	N P S U	X Z

96 / 284
le cnam
Nicolas Travers

SORT UNIQUE : Exercice $|\mathcal{M}| = 3$

- On souhaite projeter et éliminer les doublons de la seconde colonne de la table (*lettre*)
- Capacité après projection : 1 page $\Rightarrow$ 4 lettres

1 A 5 2 B 7 3 L 3	4 C 1 5 H 4 6 E 3	7 A 1 8 E 3 9 X 5	10 C 1 11 F 4 12 G 6	13 D 0 14 X 8 15 Y 0	16 Z 4 17 H 2 18 B 6	19 L 1 20 K 0 21 G 3	22 B 6 23 C 2 24 P 6	25 U 3 26 N 4 27 U 6	28 Z 7 29 H 0 30 I 5	31 A 6 32 L 9 33 S 4	34 A 3 35 X 0 36 Z 1

Table en mémoire :

Stockage temporaire
1) Phase de Tri

A B C E	F H L X	B D G H	L X Y Z	B C G K	N P U Z	A H I L	S X Z

2) Phase de Fusion

A B C D	E F G H	L X Y Z	A B C G	H I K L	N P S U	X Z	

2) Phase de Fusion

A B C D	E F G H	I K L N	P S U X	Y Z

SORT UNIQUE : Complexité

- Accès à $\mathcal{R}$: $\Rightarrow \mathcal{R}'$
- Écriture des blocs triés sans doublons : $|\mathcal{R}'|$
- Tri fusion des blocs : $2 \times |\mathcal{R}'| \left\lceil \log_{|\mathcal{M}|-1}(|\mathcal{R}'|) \right\rceil$
- Coût global : Accès à $\mathcal{R} + |\mathcal{R}'| + 2 \times |\mathcal{R}'| \times \left\lceil \log_{|\mathcal{M}|-1} |\mathcal{R}'| \right\rceil$

Projection : Exercice

- Exercice avec le même jeu de données avec $|\mathcal{M}| = 4$
- Réaliser `HASH UNIQUE`
- Réaliser `SORT AGGREGATE`

Jointures

- Plusieurs algorithmes de jointures
- Présentation d'algorithmes d'égalité
- Certains ne marchent pas en inégalité
- Tables $\mathcal{R}$ et $\mathcal{S}$
- Jointure naturelle sur l'attribut a
- $|\mathcal{M}|$ pages pour l'opération de jointure

Jointures : 4 algorithmes

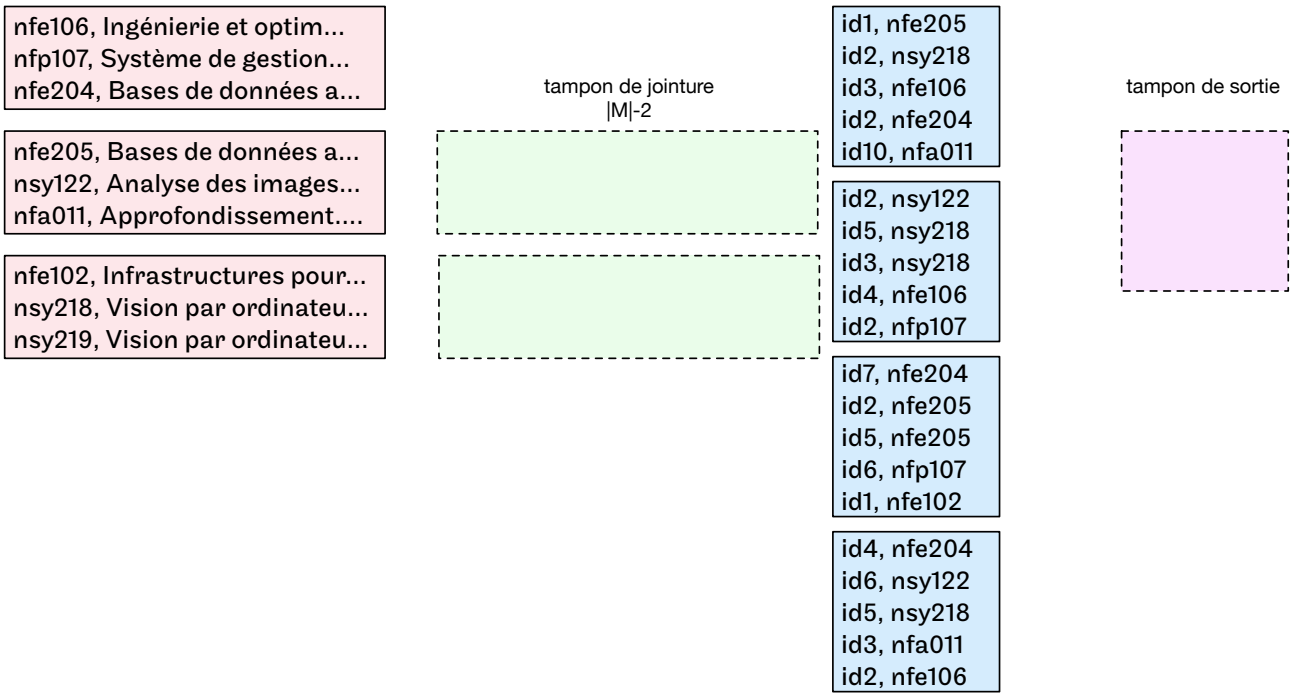
- Jointure par **Boucles Imbriquées**
  - Simple : `NESTED LOOPS` (NLJ)
  - Avec index : `NESTED LOOPS + INDEX RANGE SCAN` (NLJI)
- Jointure par **Tri Fusion**
  - `MERGE JOIN + SORT JOIN` (SMJ)
- Jointure par **Hachage** :
  - `HASH JOIN`
  - Tiens en mémoire : *MemoryHashJoin* (MHJ)
  - Sinon : *GraceHashJoin* (GHJ)

NESTED LOOPS : Algorithme

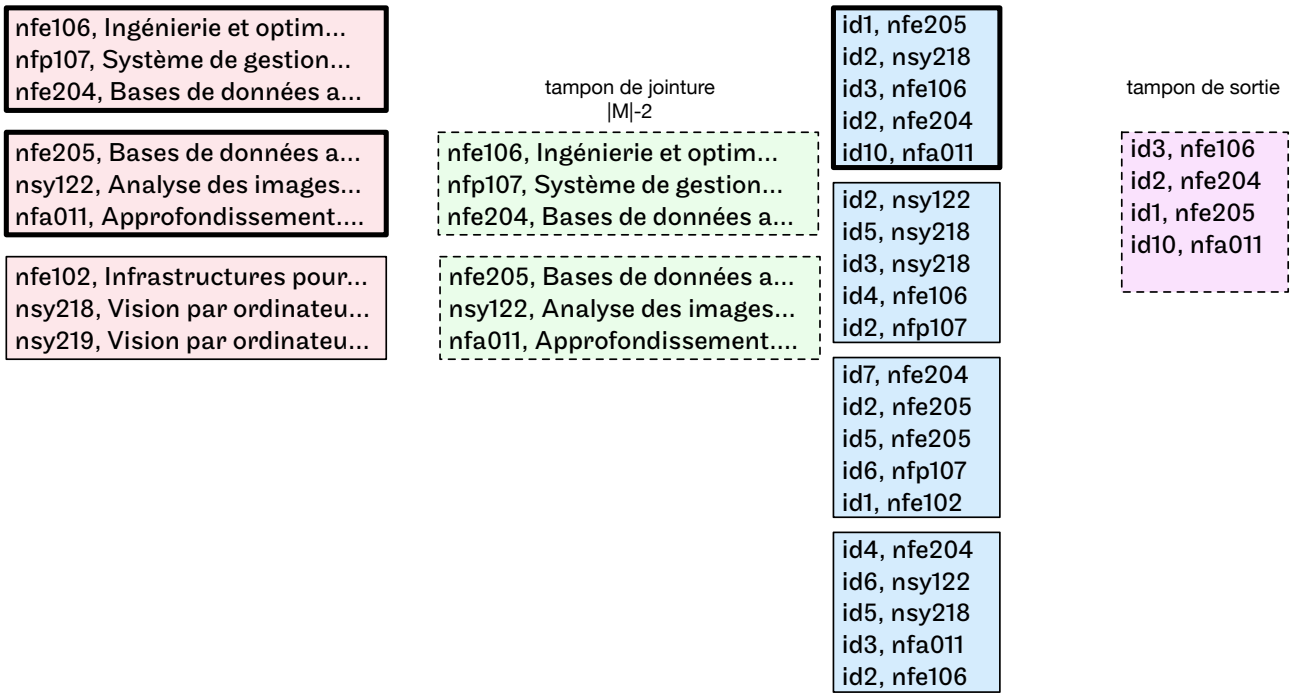
- Trois tampons nécessaires :
  - 1 tampon pour le résultat de la jointure $\mathcal{T}$
  - 1 tampon pour lire séquentiellement $\mathcal{S}$
  - $|\mathcal{M}| - 2$ pour les données de $\mathcal{R}$
- Algorithme :
  - Pour chaque bloc de $|\mathcal{M}| - 2$ provenant de $\mathcal{R}$
    - Pour chaque page de $\mathcal{S}$

      Joindre en mémoire les tuples
      des deux pages sélectionnées dans $\mathcal{T}$

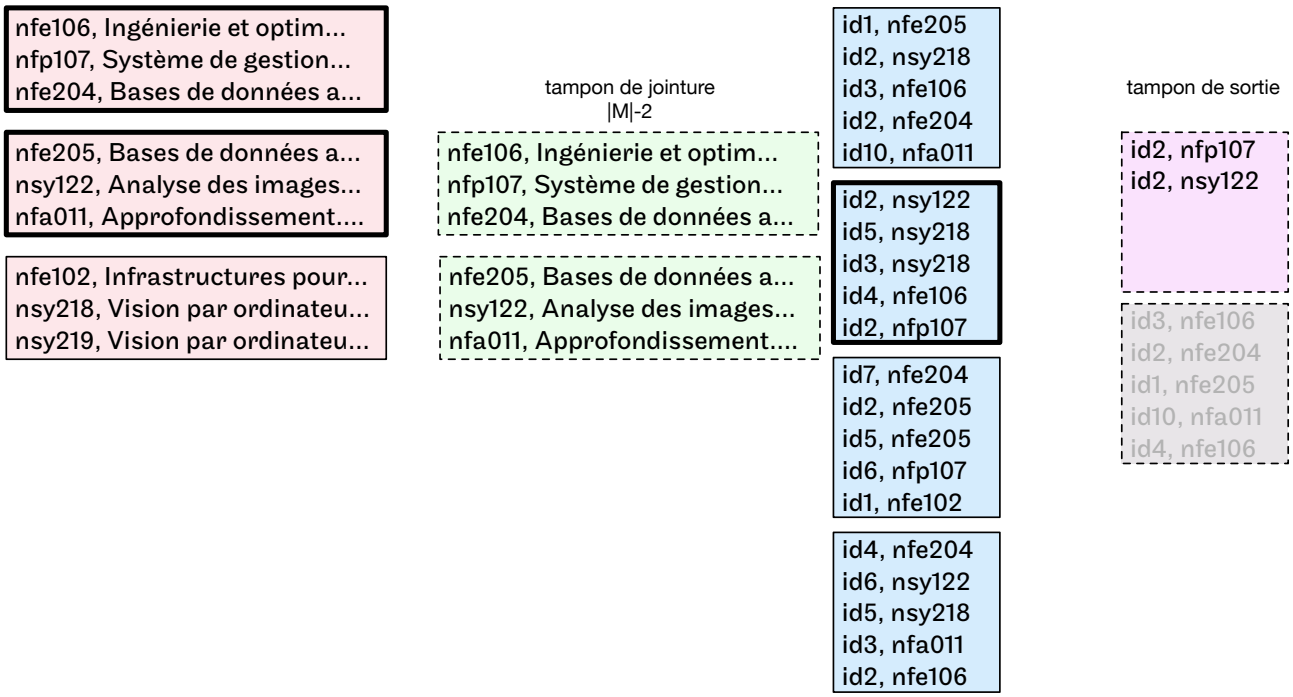
NESTED LOOPS : Exemple ($|\mathcal{M}| = 4$)



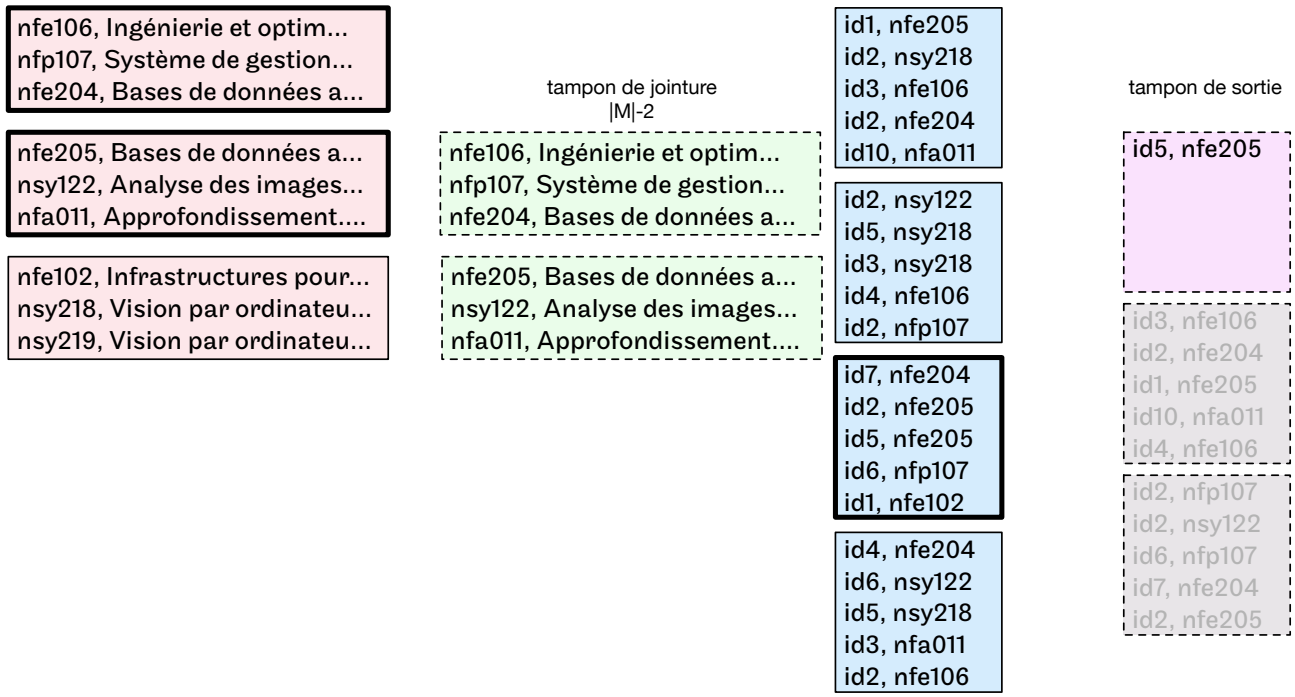
NESTED LOOPS : Exemple ($|\mathcal{M}| = 4$)



NESTED LOOPS : Exemple ($|\mathcal{M}| = 4$)

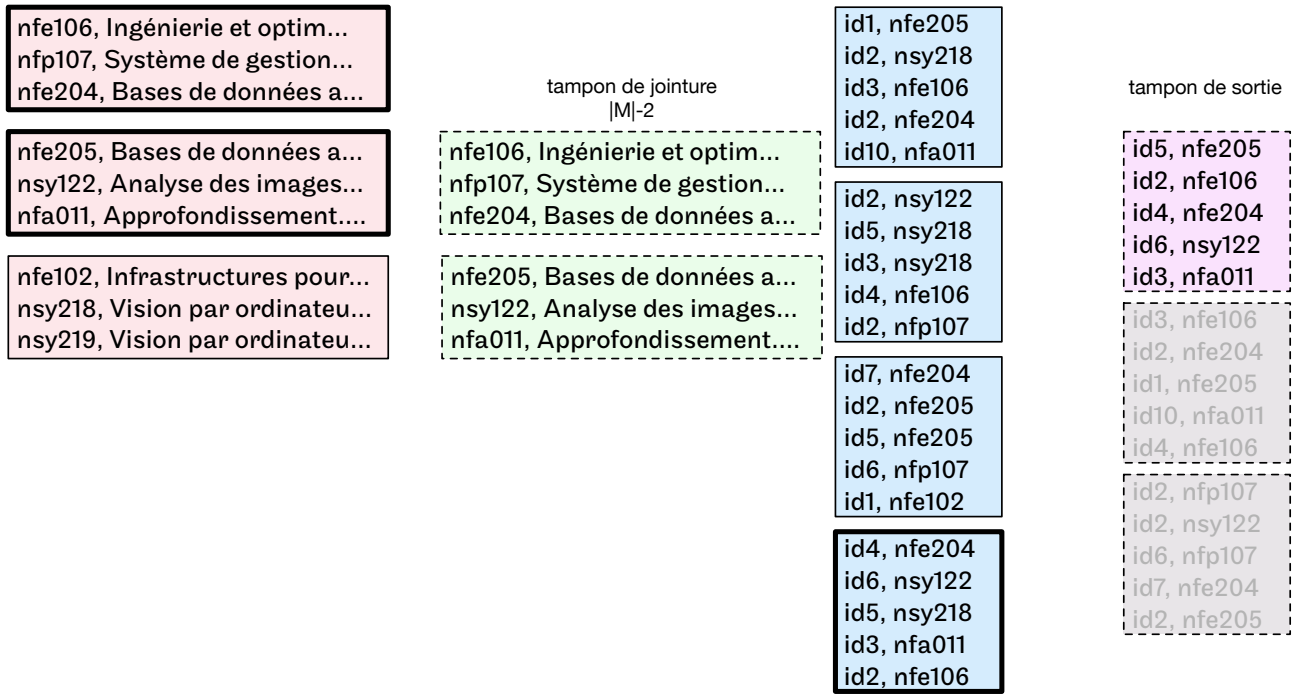


NESTED LOOPS : Exemple ($|\mathcal{M}| = 4$)

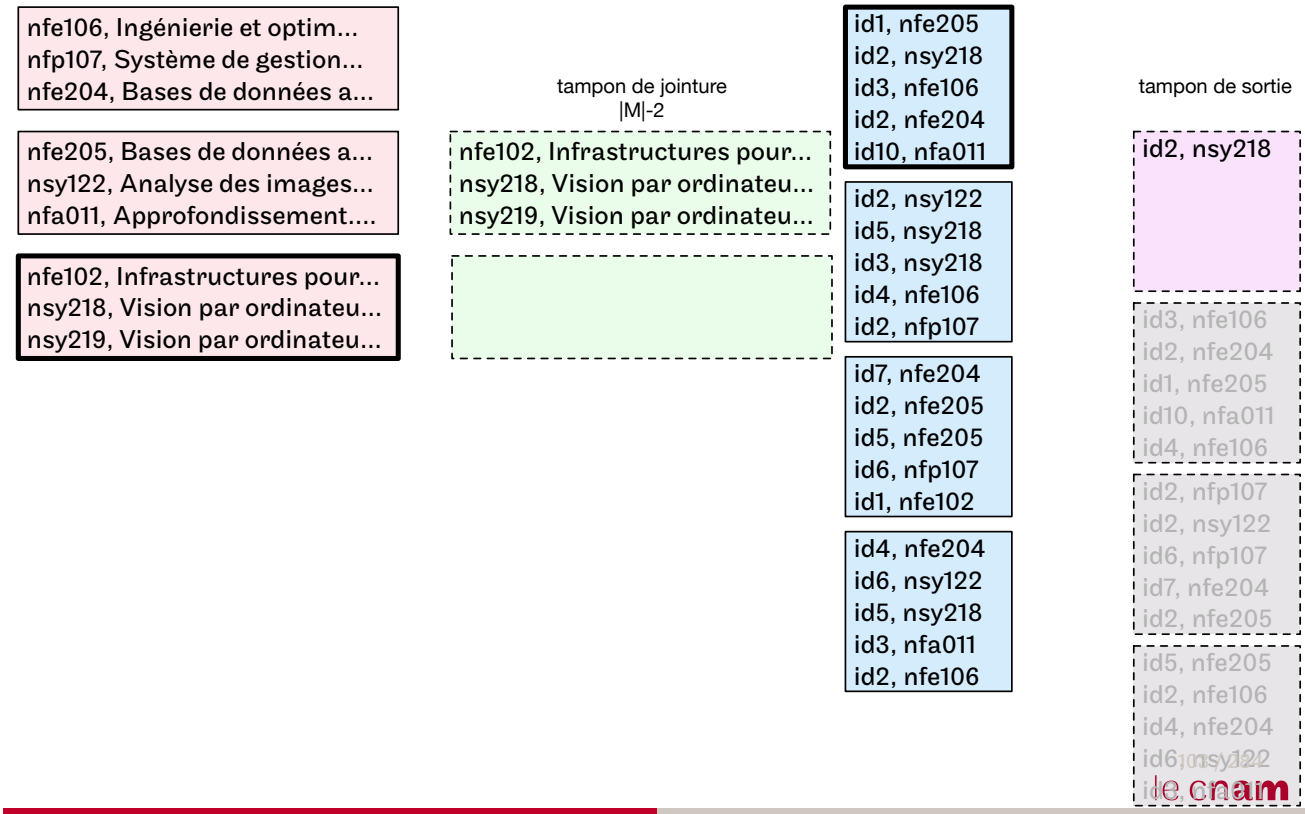


Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

NESTED LOOPS : Exemple ($|\mathcal{M}| = 4$)

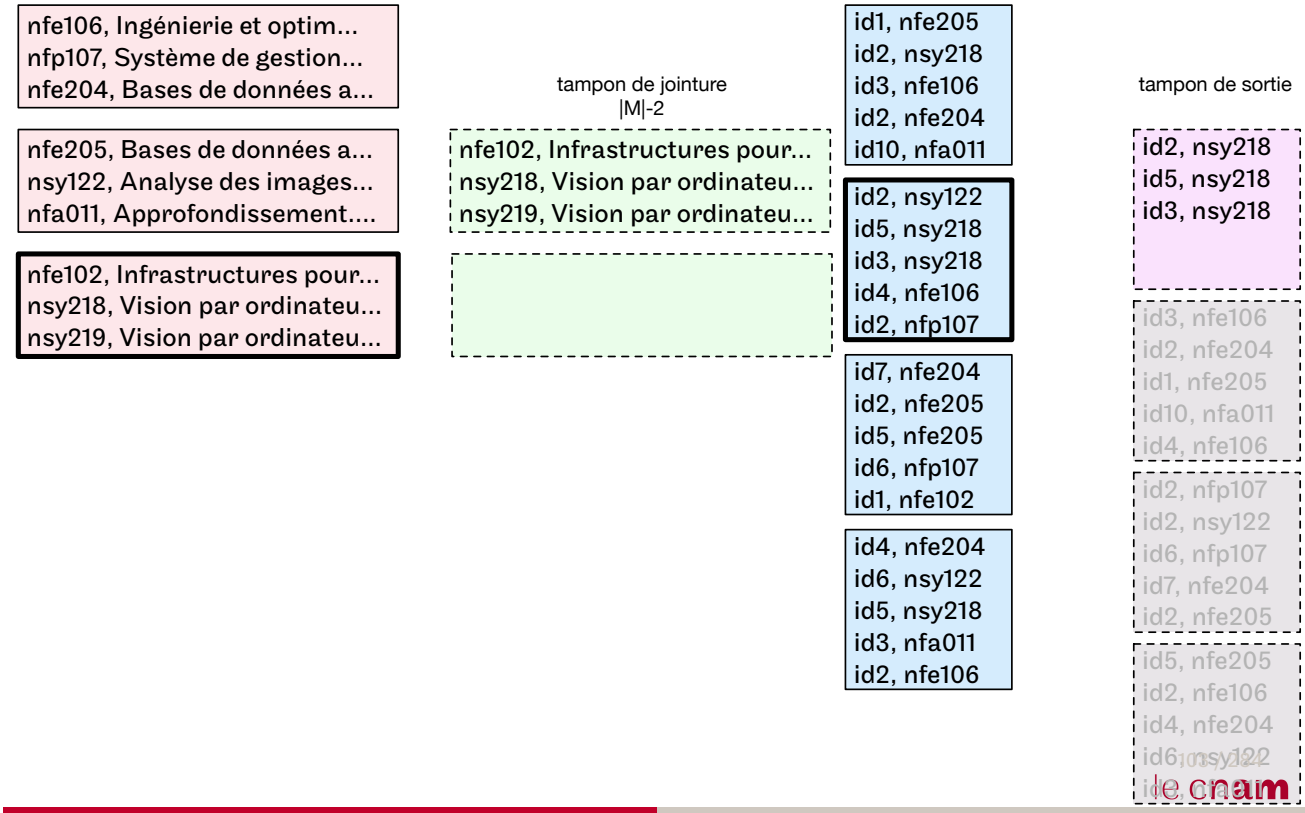


NESTED LOOPS : Exemple ($|\mathcal{M}| = 4$)



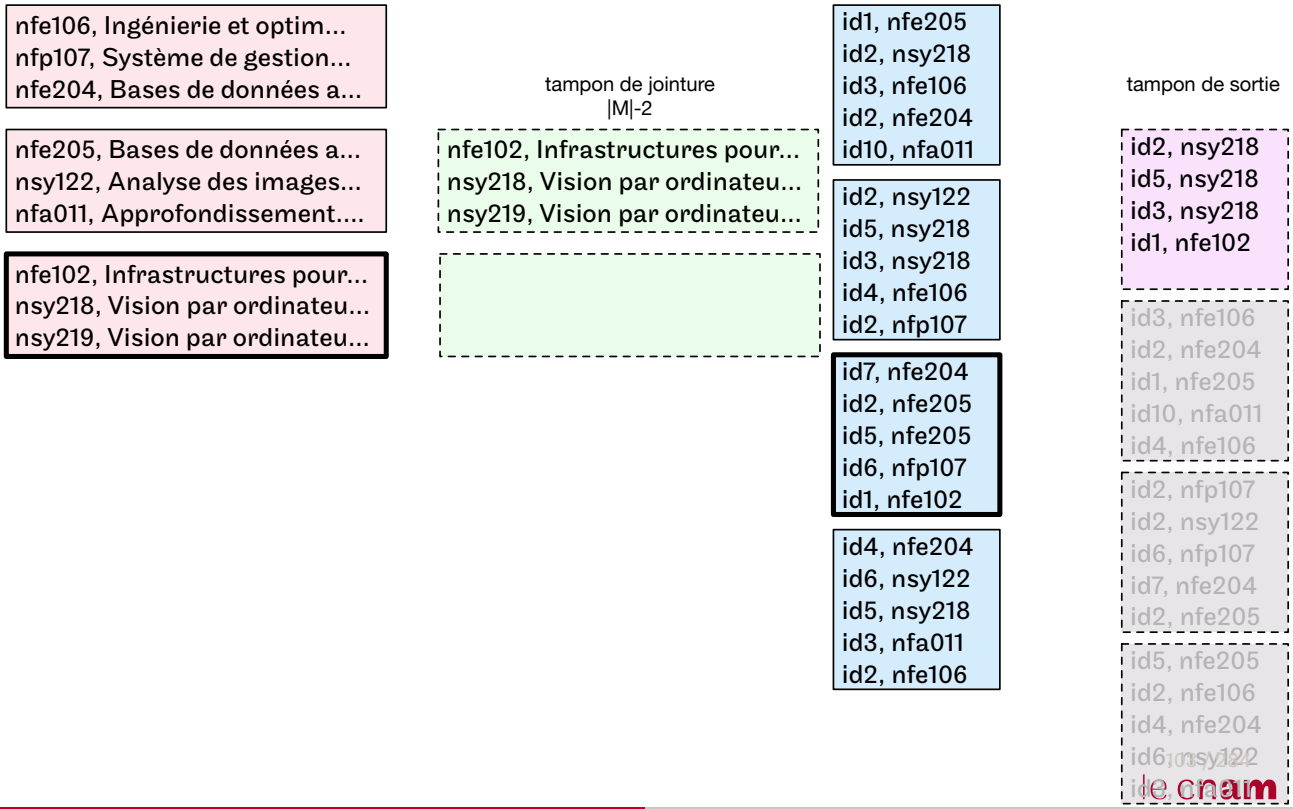
Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

NESTED LOOPS : Exemple ($|\mathcal{M}| = 4$)



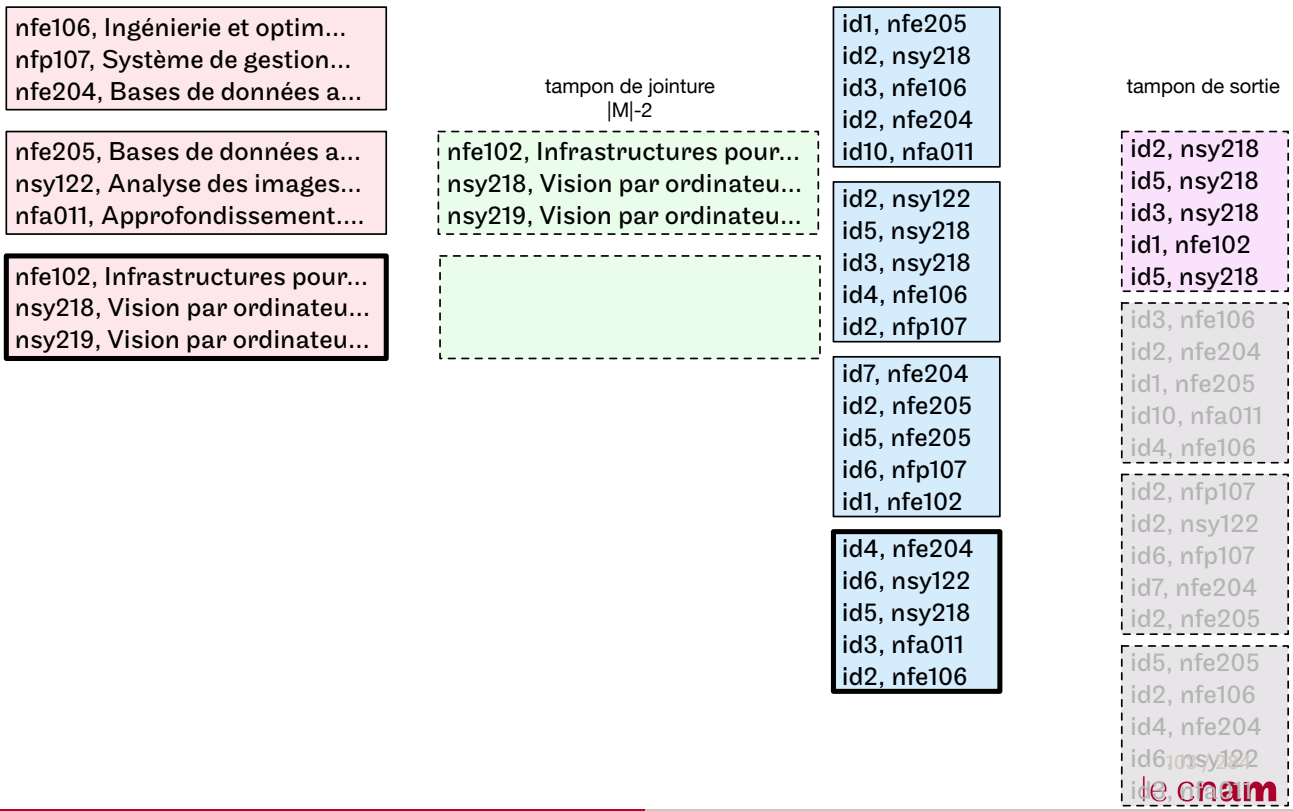
Nicolas Travers

NESTED LOOPS : Exemple ($|\mathcal{M}| = 4$)



Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

NESTED LOOPS : Exemple ($|\mathcal{M}| = 4$)



Nicolas Travers

NestedLoopJoin : Complexité

- Accès à $\mathcal{R}$: Table Access Full/By rowids, Partition Hash
  Ici SelSeq : $|\mathcal{R}| \Rightarrow |\mathcal{R}'|$
- Lecture séquentielle de $\mathcal{S}$: $\left\lceil \frac{|\mathcal{R}'|}{|\mathcal{M}|-2} \right\rceil \times |\mathcal{S}|$
- Coût de l'opération : Accès à $\mathcal{R} + \left\lceil \frac{|\mathcal{R}'|}{|\mathcal{M}|-2} \right\rceil \times |\mathcal{S}|$
- Coûte cher si les tables sont grosses

NESTED LOOPS + INDEX RANGE SCAN : Algorithme

- 1 Tampon pour les n-uplets de $\mathcal{R}$
- 1 Tampon pour l'index $\mathcal{I}$ (sur attribut de jointure)
- 1 Tampon pour les accès à $\mathcal{S}$
- Pour chaque n-uplet t de $\mathcal{R}$

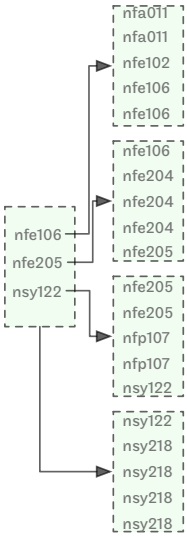
  Accès à $\mathcal{I}$ avec $t.a$
  Tri des ROWIDs (clustering factor)
  Accès aux pages pointées

NESTED LOOPS + INDEX RANGE SCAN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



liste de ROWID

id1, nfe205
id2, nsy218
id3, nfe106
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, nfe106
id2, nfp107

id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

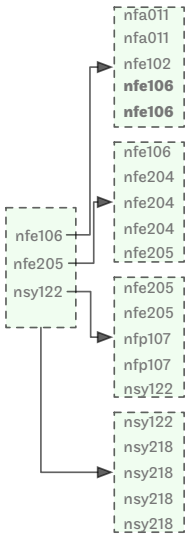
tampon de sortie

NESTED LOOPS + INDEX RANGE SCAN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



liste de ROWID

@1.3
@2.4

id1, nfe205
id2, nsy218
id3, **nfe106**
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, **nfe106**
id2, nfp107

id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

tampon de sortie

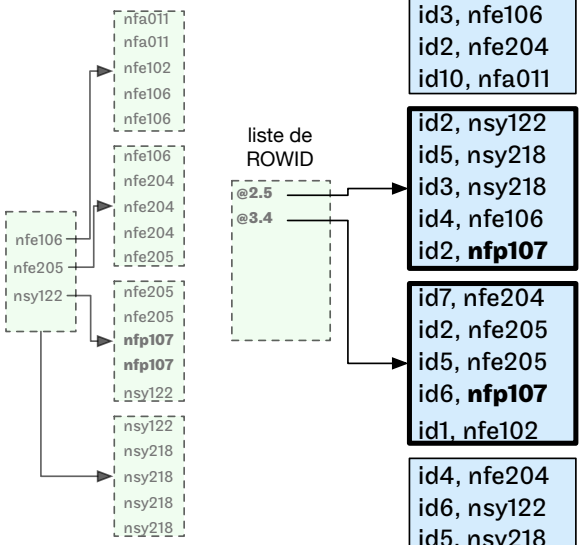
id3, nfe106
id4, nfe106

NESTED LOOPS + INDEX RANGE SCAN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



tampon de sortie

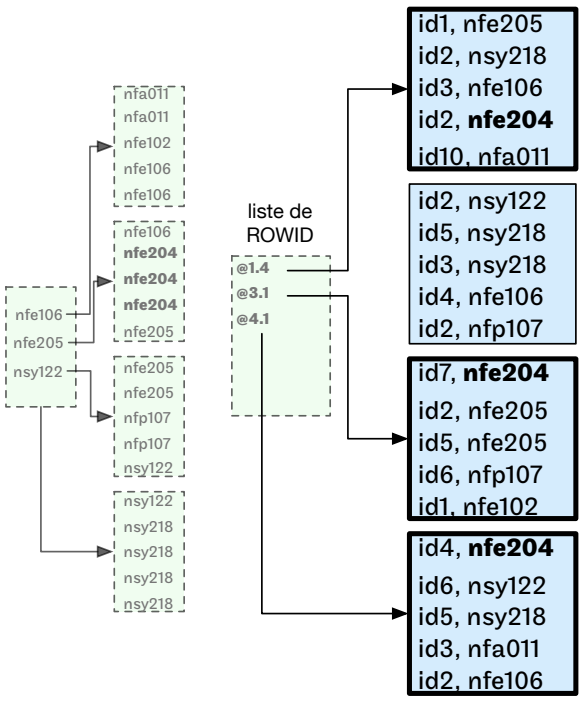
id3, nfe106
id4, nfe106
id2, nfp107
id6, nfp107

NESTED LOOPS + INDEX RANGE SCAN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...

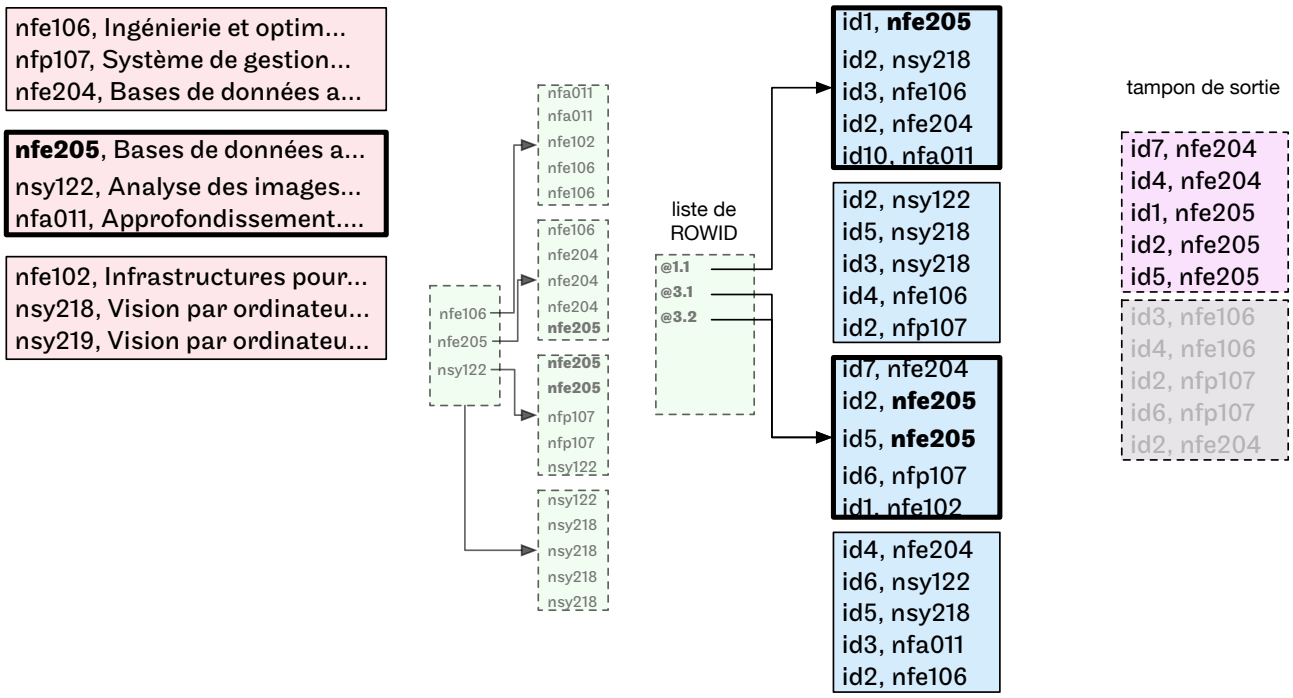


tampon de sortie

id7, nfe204
id4, nfe204

id3, nfe106
id4, nfe106
id2, nfp107
id6, nfp107
id2, nfe204

NESTED LOOPS + INDEX RANGE SCAN : Exemple



NESTED LOOPS + INDEX RANGE SCAN : Complexité

- Accès à $\mathcal{R} + ||\mathcal{R}'|| \times (|\mathcal{I}| + \Delta + \phi \times Sel)$ [18]
  - Coût index unique : $Sel = \frac{1}{||\mathcal{S}||}$, $\Delta = 0$
  - Coût index non dense : $\phi = |\mathcal{R}|$
- $||\mathcal{R}'||$ = nombre de tuples en sortie de "Accès à $\mathcal{R}$"
- Sélectivité de la jointure : $\frac{1}{||\mathcal{R}||}$ (en moyenne [19])
- Coût de traversée de l'index à chaque n-uplet de $\mathcal{R}$

18. Formule d'accès via index
19. Pas d'arrondis

MERGE JOIN + SORT JOIN : Jointure par Tri-Fusion

- SORT JOIN : Projection et Tri de $\mathcal{R}$
- SORT JOIN : Projection et Tri de $\mathcal{S}$
- MERGE JOIN : Fusion des deux listes de valeurs
- Que doit-on projeter pour l'expression suivante : $\pi_{\mathcal{R}.a,\mathcal{S}.b}(\mathcal{R} \bowtie_{\mathcal{R}.x=\mathcal{S}.x} \mathcal{S})$

MERGE JOIN + SORT JOIN : Complexité

- Tri [20] :
  - Accès à $\mathcal{R} + |\mathcal{R}'| + 2 \times |\mathcal{R}'| \times \left\lceil \log_{|\mathcal{M}|-1}(|\mathcal{R}'|) \right\rceil$
  - Accès à $\mathcal{S} + |\mathcal{S}'| + 2 \times |\mathcal{S}'| \times \left\lceil \log_{|\mathcal{M}|-1}(|\mathcal{S}'|) \right\rceil$
- Fusion : $|\mathcal{R}'| + |\mathcal{S}'|$
- Forte domination du tri dans la jointure
- Utile pour :
  - Tables non indexées
  - Tables déjà triées (suppression du coût de tri)
  - Elimination des duplicats ou tri

20. Formule du Tri-Fusion

HASH JOIN : Join par Hachage

- Jointures par égalité (équi-jointures) et Jointures naturelles
- Deux possibilités :
  1. Une table projetée peut tenir en mémoire : **in Memory Hash Join**
  2. Une table projetée *et partitionnée*, peut tenir en mémoire : **Grace Hash Join**

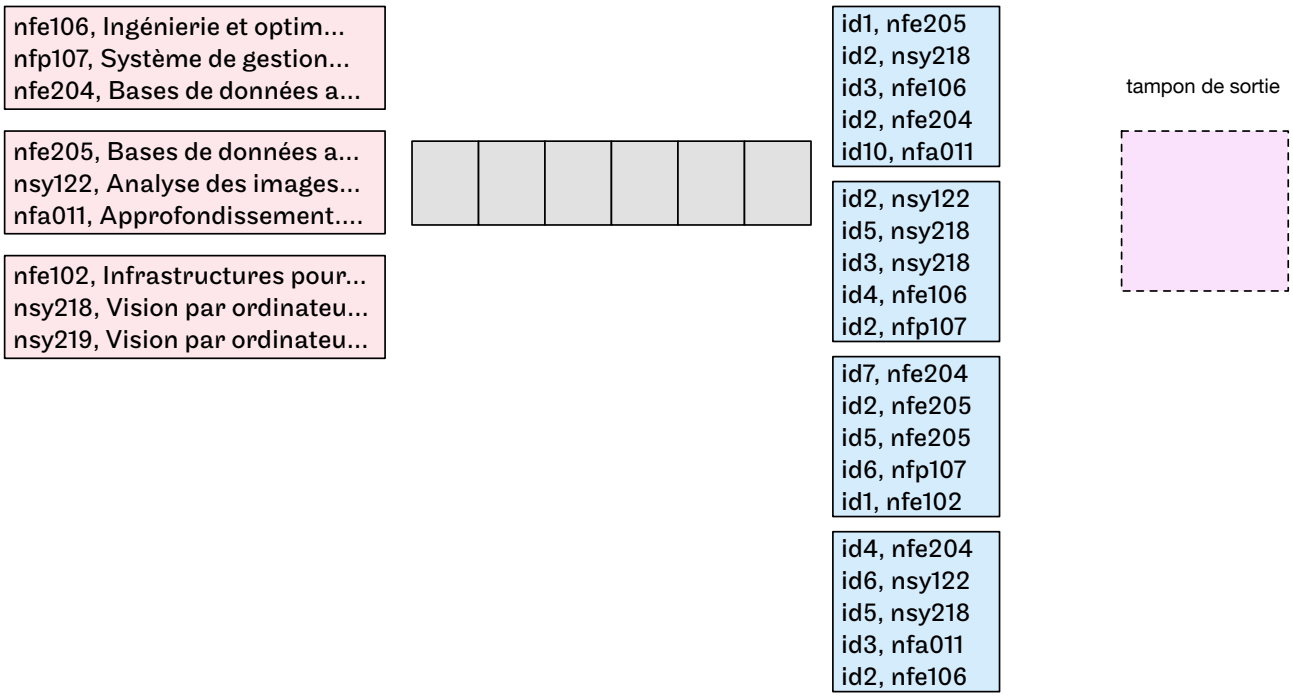
HASH JOIN : MHJ - Jointure en mémoire

- $|\mathcal{R}'| < |\mathcal{M}|$ (généralement *left-join*)
- $|\mathcal{M}| - 2$ partitions (1 tampon en lecture, 1 tampon pour la sortie)
- Principe :
  1. Accès à $\mathcal{R}$, puis partitionnement de $\mathcal{R}$ sur l'attribut de jointure
  2. Accès à $\mathcal{S}$, puis jointure avec le hachage de $\mathcal{R}$

HASH JOIN : MHJ - Algorithme

- Accès à $\mathcal{R}$
- Pour chaque n-uplet t de $\mathcal{R}'$
  - Placer $\pi_{a,x}(t)$ dans $h(t.a)$
- Accès à $\mathcal{S}$
- Pour chaque n-uplet t de $\mathcal{S}'$
  - Vérifier si $t.a$ existe dans la partition $h(t.a)$
  - Si oui, joindre les tuples

MEMORY HASH JOIN : Exemple

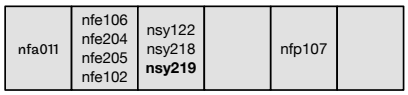


MEMORY HASH JOIN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



id1, nfe205
id2, nsy218
id3, nfe106
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, nfe106
id2, nfp107

id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

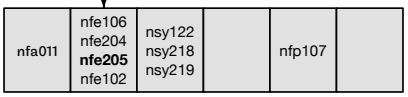
tampon de sortie

MEMORY HASH JOIN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



id1, **nfe205**
id2, nsy218
id3, nfe106
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, nfe106
id2, nfp107

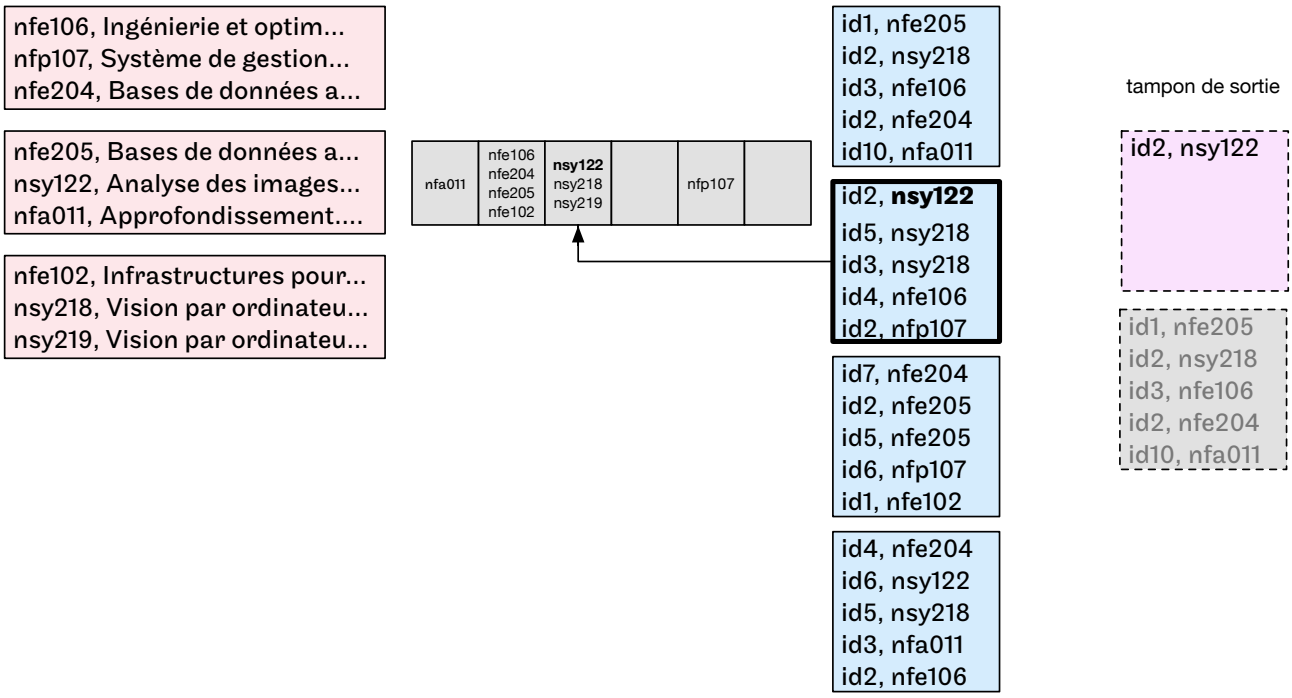
id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

tampon de sortie

id1, nfe205

MEMORY HASH JOIN : Exemple



HASH JOIN : MHJ - Complexité

- Hachage $\mathcal{R}$: Accès à $\mathcal{R}$
- Jointure : Accès à $\mathcal{S}$
- Coût total : Accès à $\mathcal{R}$ + Accès à $\mathcal{S}$
- Limite d'utilisation : $|\mathcal{R}'| < |\mathcal{M}| - 1$

HASH JOIN : GHJ - Jointure partiellement en mémoire

- Si pour $\mathcal{R}$ et $\mathcal{S}$:
  - Supérieure à $|\mathcal{M}| - 1$ (après projection)
  - Découpé en $|\mathcal{M}| - 1$ partitions
  - Taille max d'une partition $|\mathcal{M}| - 2$
- Algorithme :
  1. Partitionnement de $\mathcal{R}$ et $\mathcal{S}$ sur l'attribut de jointure ($|\mathcal{M}| - 1$ partitions)
  2. NESTED LOOPS sur chaque partition de même valeur de hachage

HASH JOIN : GHJ - Algorithme

- Besoin de $|\mathcal{M}| - 1$ tampon pour le hachage
- Pour chaque n-uplet t de $\mathcal{R}'$ ($|\mathcal{R}'|$ et $|\mathcal{S}'|$)
  - Si tampon de h(t.a) plein $\Rightarrow$ vider dans la partition $\mathcal{R}|h(t.a)$
  - Placer $\pi_{a,x}(t)$ dans $h(t.a)$
- Pour chaque couple i de partition
  - Faire jointure par boucle imbriquée $NLJ(\mathcal{P}_{Ri}, \mathcal{P}_{Si})$

GRACE HASH JOIN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...

id1, nfe205
id2, nsy218
id3, nfe106
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, nfe106
id2, nfp107

id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

tampon de sortie

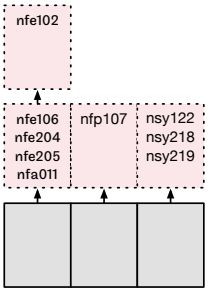
Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

GRACE HASH JOIN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



id1, nfe205
id2, nsy218
id3, nfe106
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, nfe106
id2, nfp107

id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

tampon de sortie

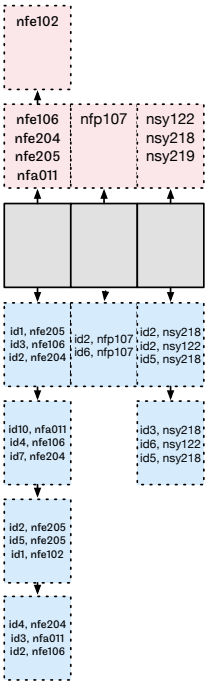
117 / 284
le cnam
Nicolas Travers

GRACE HASH JOIN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



id1, nfe205
id2, nsy218
id3, nfe106
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, nfe106
id2, nfp107

id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

tampon de sortie

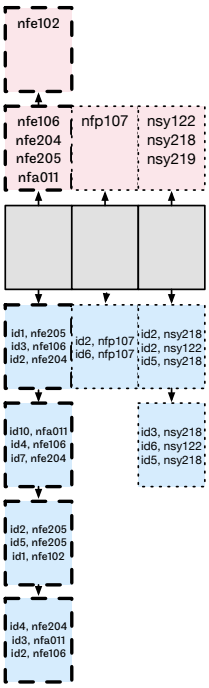
Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

GRACE HASH JOIN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



id1, nfe205
id2, nsy218
id3, nfe106
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, nfe106
id2, nfp107

id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

tampon de sortie

id3, nfa011
id2, nfe106

id3, nfe106
id2, nfe204
id1, nfe205
id4, nfe106
id7, nfe204

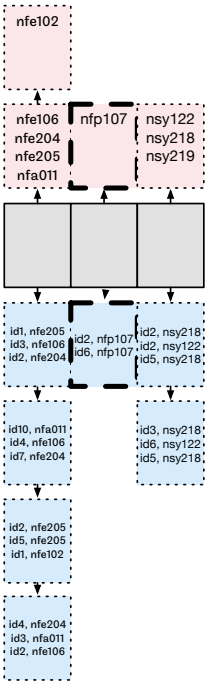
id10, nfa011
id1, nfe102
id2, nfe205
id5, nfe205
id4, nfe204

GRACE HASH JOIN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



id1, nfe205
id2, nsy218
id3, nfe106
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, nfe106
id2, nfp107

id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

tampon de sortie

id3, nfa011
id2, nfe106
id2, nfp107
id6, nfp107

id3, nfe106
id2, nfe204
id1, nfe205
id4, nfe106
id7, nfe204

id10, nfa011
id1, nfe102
id2, nfe205
id5, nfe205
id4, nfe204

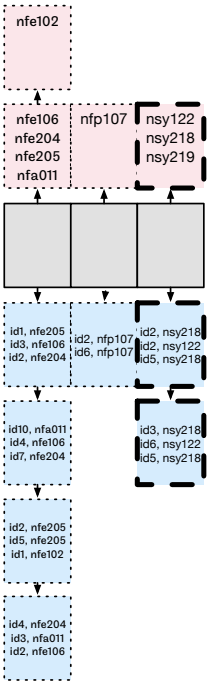
Rappels Opérations Plans SGBD Tuning Intro Indexes Opérations Sel Sort Proj Join

GRACE HASH JOIN : Exemple

nfe106, Ingénierie et optim...
nfp107, Système de gestion...
nfe204, Bases de données a...

nfe205, Bases de données a...
nsy122, Analyse des images...
nfa011, Approfondissement....

nfe102, Infrastructures pour...
nsy218, Vision par ordinateu...
nsy219, Vision par ordinateu...



id1, nfe205
id2, nsy218
id3, nfe106
id2, nfe204
id10, nfa011

id2, nsy122
id5, nsy218
id3, nsy218
id4, nfe106
id2, nfp107

id7, nfe204
id2, nfe205
id5, nfe205
id6, nfp107
id1, nfe102

id4, nfe204
id6, nsy122
id5, nsy218
id3, nfa011
id2, nfe106

tampon de sortie

id2, nsy218
id5, nsy218
id6, nsy122
id3, nsy218
id5, nsy218

id3, nfe106
id2, nfe204
id1, nfe205
id4, nfe106
id7, nfe204

id10, nfa011
id1, nfe102
id2, nfe205
id5, nfe205
id4, nfe204

id3, nfa011

HASH JOIN : GHJ - Complexité

- Hachage : Accès à $\mathcal{R} + |\mathcal{R}'|$+ Accès à $\mathcal{S} + |\mathcal{S}'|$
- Taille d'un partition : $\frac{|\mathcal{R}'|}{|\mathcal{M}|-1}$
- Jointures : $(|\mathcal{M}| - 1) \times (\frac{|\mathcal{R}'|}{|\mathcal{M}|-1} + \frac{|\mathcal{S}'|}{|\mathcal{M}|-1}) \Rightarrow |\mathcal{R}'| + |\mathcal{S}'|$
- Coût total : Accès à $\mathcal{R} + 2 \times |\mathcal{R}'|$+ Accès à $\mathcal{S} + 2 \times |\mathcal{S}'|$
- Limite d'utilisation :
  $|\mathcal{R}'| \leq (|\mathcal{M}| - 1)(|\mathcal{M}| - 2) \Rightarrow |\mathcal{R}'| < |\mathcal{M}|^2$

Jointures : Rappels

1. `NESTED LOOPS` : Une des tables est petite (projection ?)
2. `NESTED LOOPS + INDEX RANGE SCAN` : Dépend des statistiques
3. `MERGE JOIN + SORT JOIN` : Une relation est ou doit être triée
4. `HASH JOIN` : MHJ - Partitionnement d'une table
5. `HASH JOIN` : GHJ - Partitionnement des deux tables

Introduction

- **SQL est déclaratif**
  - On dit ce que l'on veut obtenir
  - On ne dit pas comment l'obtenir
- **Optimiseur** doit :
  - Comprendre la requête. Traduction en **P***lan d'***E***xécution* **L***ogique*[21] (PEL)
  - Choisir le meilleur **P***lan d'***E***xécution* **P***hysique*[22] (PEP)
  - Exécuter le PEP

21. Opérateurs de l'algèbre relationnelle
22. Opérateurs implémentés dans le SGBD

Introduction : Exemple

- Soit le schéma :
  - *Cours* (CODE, Intitule, Responsable)
  - *Auditeurs* (CODE_UE, CODE_AUDITEUR, Annee, Note)
- Soit la requête :
  - Liste des *années* où *Nicolas Travers* a eu des auditeurs ?

⇒ SELECT Annee
FROM Cours, Auditeurs
WHERE Responsable = 'Nicolas Travers' AND
CODE = CODE_UE ;

Introduction : Exemple PEL

1. $\pi_{Annee}(\sigma_{Responsable='NicolasTravers'}(Cours \bowtie_{CODE=CODE\_UE} Auditeurs))$
2. $\pi_{Annee}(\sigma_{Responsable='NicolasTravers'}(Cours) \bowtie_{CODE=CODE\_UE} Auditeurs)$
3. $\pi_{Annee}(\sigma_{Responsable='NicolasTravers'}(Auditeurs \bowtie_{CODE=CODE\_UE} Cours))$
4. $\pi_{Annee}(Auditeurs \bowtie_{CODE=CODE\_UE} \sigma_{Responsable='NicolasTravers'}(Cours))$

Quel est le meilleur plan d'exécution ?

⇒ Le second PEL

Rappel sur les plans d'exécution

- Deux étapes pour l'optimiseur
  1. Traduction de la requête dans PEL
  2. Traduction du PEL en PEP

PEL : Décomposition de la requête

- Requête SQL décomposée en blocs
  - `Select From Where`
  - Une seule clause `Group By` et `Having`
- Optimisation bloc par bloc
- Requête imbriquée ⇒ groupe de blocs imbriqué

PEL : Exemple d'imbrication

```
SELECT intitule, COUNT(*)
FROM Cours, Auditeur
WHERE CODE_UE = CODE
    AND Responsable = 'Nicolas Travers'
GROUP BY intitule
HAVING COUNT(*) > (SELECT AVG(COUNT(*))
                   FROM Cours, Auditeur
                   WHERE CODE_UE = CODE
                   GROUP BY CODE)
```

Forme Normale Conjonctive

- Toute requête peut être mise sous forme normale conjonctive (FNC) :
  (A=a ∪ B=b) ∩ (A= a ∪ C=c)
- Optimisation de A=a ∪ B=b (blocs indépendants)
  - Si index sur A et index sur B
    ⇒ Traverser les deux index et faire l'union
  - Sinon, balayage séquentiel

PEL : Exemple de plan

- SELECT intitule
  FROM Cours, Auditeur
  WHERE CODE_UE = CODE
  AND Responsable = 'Nicolas Travers'
- $\pi_{intitule}(\sigma_{responsable='NT'}(Cours \bowtie_{CODE\_UE=CODE} Auditeur))$

Règles de réécriture

1. **Commutativité de la jointure** :
   $\mathcal{R} \bowtie \mathcal{S} \equiv \mathcal{S} \bowtie \mathcal{R}$
2. **Associativité de la jointure** :
   $(\mathcal{R} \bowtie \mathcal{S}) \bowtie \mathcal{T} \equiv \mathcal{S} \bowtie (\mathcal{R} \bowtie \mathcal{T})$
3. **Regroupement des sélections** :
   $\sigma_{A='a'}(\sigma_{B='b'}\mathcal{R}) \equiv \sigma_{A='a' \wedge B='b'}\mathcal{R}$
4. **Commutativité de la sélection et de la projection** :
   $\pi_{A_1 \ldots A_n}(\sigma_{A_i='a'}\mathcal{R}) \equiv \sigma_{A_i='a'}(\pi_{A_1 \ldots A_n}\mathcal{R}), i \in 1, \ldots, n$
5. **Commutativité de la sélection et de la jointure** :
   $\sigma_{A='a'}(\mathcal{R} \bowtie \mathcal{S}) \equiv \sigma_{A='a'}(\mathcal{R}) \bowtie \mathcal{S}$
6. **Distributivité de la sélection sur l'union** :
   $\sigma_{A='a'}(\mathcal{R} \cup \mathcal{S}) \equiv \sigma_{A='a'}\mathcal{R} \cup \sigma_{A='a'}\mathcal{S}$
7. **Commutativité de la projection et de la jointure** :
   $\pi_{A_1 \ldots A_n B_1 \ldots B_m}(\mathcal{R} \bowtie_{A_i=B_j} \mathcal{S}) \equiv \pi_{A_1 \ldots 1_n}\mathcal{R} \bowtie_{A_i=B_j} \pi_{B_1 \ldots B_m}\mathcal{S}$
   $i \in 1..n, j \in 1..m$
8. **Distributivité de la projection sur l'union** :

Heuristique pour le PEL

1. Remonter les sélections le plus tôt possible (règles 4 & 5)
2. Remonter les projections le plus haut possible (règles 4, 7 & 8)
3. Décomposer les sélections en une composition. (règle 3)

$\Rightarrow \pi_{intitule}($
$\quad \pi_{CODE,intitule}(\sigma_{responsable='NT'}(Cours))$
$\quad \bowtie_{CODE\_UE=CODE}$
$\quad \pi_{CODE\_UE}(Auditeur))$

Exemple 2

- SELECT P.nom
  FROM Cours C, Auditeur A, Personne P
  WHERE C.CODE_UE = A.CODE
  AND Responsable = 'Nicolas Travers'
  AND Cursus = 'Base de Données'
  AND A.CODE_AUD = P.CODE

$\Rightarrow \pi_{nom}(\sigma_{responsable='NT' \wedge cursus='BD'}($
$Cours \bowtie_{CODE\_UE=CODE} ($
$Auditeur \bowtie_{CODE\_AUD=CODE} Personne)))$

Exemple 2 équivalent

- $\pi_{nom}($
  $\pi_{CODE}(\sigma_{responsable='NT'}(Cours))$
  $\bowtie_{CODE\_UE=CODE} ($
  $\pi_{CODE\_AUD,CODE\_UE}(Auditeur)$
  $\bowtie_{CODE\_AUD=CODE}$
  $\pi_{CODE,nom}(\sigma_{cursus='BD'}(Personne))$
  $)$
  $)$

Plan d'Exécution Physique

- Génération du PEL nécessaire mais pas suffisant
- Besoin des informations physiques
- Prendre en compte :
  - Accès disponibles (index)
  - Organisation de tables (tri, partition)
  - Statistiques (nb tuples, pages, cardinalité/sélectivité, Histogramme, clustering factor)

PEP : Caractéristiques

- Représentation arborescente
- `Feuilles` : Accès aux données
- `Noeuds internes` : Opérateur physique
- `Arc` (bas en haut) : Flot de données, consommé par l'opérateur du dessus

EXPLAIN : Outils d'affichage de PEP

- Outils de représentation de PEP sous Oracle et MySQL
- Représentation arborescente (tabulations)
- Estimation du coût d'évaluation
- Insertion de statistiques d'évaluation

EXPLAIN : Exemple

Id	Opération	Name
0	SELECT STATEMENT	
1	TABLE ACCESS BY INDEX ROWIDS	Cours
2 *	INDEX RANGE SCAN	Cours_NB_AUDITEUR_IDX

(2) : access(NB_AUDITEUR BETWEEN 20 AND 30)

Table EXPLAIN PLAN (Oracle)

- Toutes les données sont stockées ⇒ Plans `EXPLAIN` aussi

Oracle Commande de transformation SQL ⇒ `EXPLAIN` :

- EXPLAIN PLAN [SET STATEMENT_ID = 'ID'][23]
  [INTO < table_name >][24]
  FOR < Requête SQL >
- Commande d'affichage automatique du plan :
  `SET AUTOTRACE ON`

MySQL Commande d'affichage de EXPLAIN : `EXPLAIN` < Requête SQL >

23. Identifiant pour retrouver le plan

Sélection / Accès aux données

Opération	Option	Description

`TABLE ACCESS`	`FULL`
	`BY INDEX ROWID`
	`CLUSTER`
		Sélection séquentielle
		Accès aux ROWIDS
		Accès via la clé d'index de cluster

Accès à un Index

Opération	Option	Description
INDEX	UNIQUE SCAN	clé primaire/unique
	RANGE SCAN	données multivaluées
	FULL SCAN	toutes les clés de l'index
	FAST FULL SCAN	toutes les clés de l'index sans prendre les ROWIDS
	FULL SCAN (MIN/MAX)	min/max des valeurs de l'index

Accès à un Index Hash

Opération	Option	Description
PARTITION HASH	SINGLE	Accès à la partition sélectionné par la clé d'accès (sur le FULL SCAN)
	ALL	Accès à toutes les partitions (pas de filtrage sur la clé hachée)

Accès à un Index Bitmap

Opération	Option	Description
BITMAP	INDEX SINGLE VALUE	Accès à l'index bitmap, retourne les numéros des tuples
	CONVERSION COUNT	Compte le nombre de bit à 1
	CONVERSION TO ROWIDS	Converti les numéros de tuples en ROWIDs
	AND	ET logique entre deux listes bitmap
	OR	OU logique entre deux listes bitmap
	MINUS	AND NOT logique
	INDEX RANGE SCAN MERGE	Accès à un ensemble de listes Fusionne les listes retournées par un RANGE SCAN

Tri vs Hachage

Opération	Option	Description
SORT	ORDER BY	*Sort* pour les résultats
	AGGREGATE	Tri avec fonction d'aggrégat [25]
	GROUP BY	Tri sans fonction (juste Group By)
	UNIQUE	DISTINCT
	JOIN	Étape de tri de l'algo de jointure

Opération	Option	Description
HASH	AGGREGATE	Hachage avec fonction d'aggrégat
	GROUP BY	Hachage sans fonction
	UNIQUE	DISTINCT

25. avec ou sans Group By

Jointures

Opération	Description
`NESTED LOOPS`	*NLJ*/*NLJI*, Boucles imbriquées
`MERGE JOIN`	*SMJ*, Etape de fusion des tables
`HASH JOIN`	*HashJoin*, construction des partitions, puis suite de *NLJ*
`AND-EQUAL`	Intersection de deux listes de ROWIDs pour une jointure entre deux indexes

Options : `OUTER, ANTI, SEMI, CARTESIAN`.

Bonus (1/3)

Opération	Description
`CONNECT BY`	Auto-jointure hiérarchique utilisant le résultat de l'étape précédente
`CONCATENATION`	Union multiples de *Result Sets* (cf `UNION`)
`COUNT`	Compte le nombre tuples d'un résultat
`FILTER`	Opération de sélection
`FIRST ROW`	Premier tuple sélectionné par la requête
`FOR UPDATE`	Blocage des tuples retournés pour des MAJ ultérieures

Bonus (2/3)

Opération	Description
`VIEW`	Accès à une vue, ou création d'une table temporaire
`UNION`	Union de deux *Result Set*, sans doublons
`UNION-ALL`	Union de deux *Result Set*, avec doublons
`INTERSECTION`	Intersection de deux *Result Set*, contenant les mêmes valeurs
`MINUS`	Les tuples du premier *Result Set* sont retournés, sauf ceux présents dans le second

Bonus (3/3)

Opération	Description
`SEQUENCE`	Oracle Sequence Generator (auto_increment d'ID)
`REMOTE`	Accès à une base de données distante
`INLIST OPERATOR`	Un seul opérateur à la fois. Pas de *pipeline*

Lecture de EXPLAIN

- Structure arborescente (indentation)
- L'opérateur le moins indenté est le résultat
- Le plus indenté est l'accès aux données
- Si deux indentation de même niveau, le premier est exécuté d'abord
- Exercice sur les plans suivants :
  - Requête SQL correspondante
  - Coût d'évaluation

Exemple 1 : Sélection séquentielle

Id	Opération	Name
0	SELECT STATEMENT	
1 *	TABLE ACCESS FULL	Auditeurs

(1) : access(Annee = 2017)

- SELECT ANNEE FROM Auditeurs WHERE ANNEE=2017 ;
- Statistiques : $|Auditeurs| = 1000$
- Coût : 1000 I/O

Exemple 2 : Accès via Index unique

Id	Opération	Name
0	SELECT STATEMENT	
1	TABLE ACCESS BY INDEX ROWIDS	Cours
2 *	INDEX UNIQUE SCAN	Cours_Code_IDX

(2) : access(CODE = 'NFE106')

- SELECT RESPONSABLE FROM Cours WHERE CODE='NFE106' ;
- Statistiques : $|Cours| = 20$, $||Cours|| = 600$, $|\mathcal{I}_{code}| = 2$ (dense), $ordre_{code}$=56
- Coût : $2 + 0 + 600 \times \frac{1}{600} = 3$ I/O

Exemple 3 : Accès via Index dense

Id	Opération	Name
0	SELECT STATEMENT	
1	TABLE ACCESS BY INDEX ROWIDS	Cours
2 *	INDEX RANGE SCAN	Cours_NB_AUDITEUR_IDX

(2) : access(NB_AUDITEUR BETWEEN 20 AND 30)

- SELECT RESPONSABLE FROM Cours
  WHERE NB_AUDITEUR BETWEEN 20 AND 30 ;
- Statistiques : $|Cours| = 20$, $||Cours|| = 600$, $|\mathcal{I}_{nb\_auditeur}| = 2$ (dense), $ordre_{nb\_auditeur}$=200, $Sel_{nb\_auditeur} = \frac{1}{30}$
- Coût : $2 + 0 + 600 \times \frac{1}{30} = 22$ I/O

Exemple 4 : Accès via Table de Hachage

Id	Opération	Name
0	SELECT STATEMENT	
1	PARTITION HASH SINGLE	
2 *	TABLE ACCESS FULL	Cours

(2) : access(Responsable = 'Nicolas Travers')

- SELECT CODE FROM Cours WHERE Responsable = 'Nicolas Travers' ;
- Statistiques : $|Cours| = 20$, 10 partitions
- Coût : $\left\lceil \frac{20}{10} \right\rceil = 2$ I/O

Exemple 5 : Bitmap + Conversion

Id	Opération	Name
0	SELECT STATEMENT	
1	TABLE ACCESS BY INDEX ROWID	Auditeurs
2	BITMAP CONVERSION TO ROWIDS	
3 *	BITMAP INDEX SINGLE VALUE	Auditeurs_Annee_BITMAP

(3) : access(Annee = 2017)

- SELECT CODE FROM Auditeurs WHERE ANNEE=2017 ;
- Statistiques : $|Auditeurs| = 1000$, $||Auditeurs|| = 600000$, $\phi_{annee} = 10000$, $Sel_{annee} = 3\%$ Vecteur bitmap = 2p, Conversion = 28 I/O
- Coût : $2 + 28 + 10000 \times 3\% = 330$ I/O

Exemple 6 : boucle imbriquée

Id	Opération	Name
0	SELECT STATEMENT	
1 *	NESTED LOOPS	
2	TABLE ACCESS FULL	Cours
3	TABLE ACCESS FULL	Auditeurs

(1) : filter(CODE = CODE_UE)

- SELECT * FROM Cours, Auditeurs WHERE CODE = CODE_UE ;
- Statistiques : $|Auditeurs| = 1000$, $|Cours| = 100$, $|\mathcal{M}| = 52$
- Coût : $100 + \left\lceil \frac{100}{52-2} \right\rceil \times 1000 = 2100$ I/O

Exemple 7 : Boucle imbriquée et accès via index

Id	Opération	Name
0	SELECT STATEMENT	
1 *	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	Cours
3 *	INDEX RANGE SCAN	Cours_Responsable_IDX
4	TABLE ACCESS FULL	Auditeurs

(1) : filter(CODE = CODE_UE)
(3) : access(Responsable = 'Nicolas Travers')

- SELECT ANNEE FROM Cours, Auditeurs
  WHERE CODE = CODE_UE AND Responsable = 'Nicolas Travers' ;
- Statistiques : $|\mathcal{I}_{responsable}| = 2$, $||Cours|| = 600$, $Sel_{responsable} = 0,4\%$, $\text{ordre}_{responsable} = 56$, $|Cours'| = 1$, $|Auditeurs| = 1000$, $|\mathcal{M}| = 52$
- Coût : $(2 + 0 + 600 \times 0,4\%) + \left\lceil \frac{1}{52-2} \right\rceil \times 1000 = 1004$ I/O

Exemple 8 : Boucle imbriquée avec index

Id	Opération	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	Cours
3 *	INDEX RANGE SCAN	Cours_Responsable_IDX
4	TABLE ACCESS BY INDEX ROWID	Auditeurs
5 *	INDEX RANGE SCAN	Auditeurs_CODEUE_IDX

(3) : access(Responsable = 'Nicolas Travers')
(5) : filter(CODE = CODE_UE)

- SELECT ANNEE FROM Cours, Auditeurs
  WHERE CODE = CODE_UE AND Responsable = 'Nicolas Travers' ;
- Statistiques : $|\mathcal{I}_{responsable}| = 2$, $||Cours|| = 600$, $Sel_{responsable} = 0,4\%$, $ordre_{responsable} = 56$, $|Cours'| = 1$, $|\mathcal{I}_{CODE\_UE}| = 3$, $||Auditeurs|| = 600000$, $\phi_{CODE\_UE} = 10000$, $Sel_{CODE\_UE} = \frac{1}{600}$,
- Coût :
  $(2 + 0 + 600 \times 0,4\%) + \lceil 2,4 \rceil \times (3 + 0 + 10000 \times \frac{1}{600}) = 4 + 3 \times 19,66 = 63$ I/O

Exemple 9 : Jointure par Tri-Fusion

Id	Opération	Name
0	SELECT STATEMENT	
1 *	MERGE JOIN	
2	SORT JOIN	
3	TABLE ACCESS BY INDEX ROWID	Cours
4 *	INDEX RANGE SCAN	Cours_Responsable_IDX
5	SORT JOIN	
6	TABLE ACCESS FULL	Auditeurs

(1) : filter(CODE = CODE_UE)
(4) : access(Responsable = 'Nicolas Travers')

- SELECT ANNEE FROM Cours, Auditeurs
  WHERE CODE = CODE_UE AND Responsable = 'Nicolas Travers' ;
- Statistiques : $|\mathcal{I}_{responsable}| = 2$, $||Cours|| = 600$, $Sel_{responsable} = 0,4\%$, $ordre_{responsable} = 56$, $|Cours'| = 1$, $|Auditeurs| = 1000$, $|Auditeurs'| = 500$, $|\mathcal{M}| = 51$
- Coût : $(2 + 0 + 600 \times 0,4\%) + 1 + 0$
  $+(1000) + 500 + 2 \times 500 \times \lceil \log_{51-1}(500) \rceil$
  $+1 + 500 = (5) + (3500) + 501 = 4006$ I/O

Exemple 10 : Jointure par hachage

Id	Opération	Name
0	SELECT STATEMENT	
1 *	HASH JOIN	
2	TABLE ACCESS BY INDEX ROWID	Cours
3 *	INDEX RANGE SCAN	Cours_Responsable_IDX
4	TABLE ACCESS FULL	Auditeurs

(1) : filter(CODE = CODE_UE)
(3) : access(Responsable = 'Nicolas Travers')

- SELECT ANNEE FROM Cours, Auditeurs
  WHERE CODE = CODE_UE AND Responsable = 'Nicolas Travers' ;
- Statistiques : $|\mathcal{I}_{responsable}| = 2$, $||Cours|| = 600$, $Sel_{responsable} = 0,4\%$, $\text{ordre}_{responsable} = 56$, $|Cours'| = 1$, $|Auditeurs| = 1000$, $|Auditeurs'| = 500$, $|\mathcal{M}| = 51$
- Coût : $|Cours'| < |\mathcal{M}| \Rightarrow MHJ$
  $(2 + 0 + 600 \times 0,4\%) + 1000 = 1005$ I/O

Exemples : COUNT BITMAP

Id	Opération	Name
0	SELECT STATEMENT	
1	SORT AGGREGATE	
2 *	INDEX RANGE SCAN	Cours_Responsable_IDX

(2) : access(Responsable = 'Nicolas Travers')

- SELECT COUNT(*) FROM Cours WHERE RESPONSABLE='Nicolas Travers' ;
- Statistiques : $|\mathcal{I}_{responsable}| = 2$, $||Cours|| = 600$, $Sel_{responsable} = 0,4\%$, $\text{ordre}_{responsable} = 56$
- Coût : $2 + 0 + 0 = 2$ I/O

Id	Opération	Name
0	SELECT STATEMENT	
1	SORT AGGREGATE	
2.	BITMAP CONVERSION COUNT	
3 *	BITMAP INDEX SINGLE VALUE	Auditeurs_Annee

(3) : access(Annee = 2017)

- SELECT COUNT(*) FROM Auditeurs WHERE ANNEE=2017 ;
- Statistiques : Vecteur Bitmap : 2p
- Coût : 2 I/O

EXPLAIN : Statistiques

- EXPLAIN fourni des informations statistiques
  - Nombre d'appels physiques
  - Nombre d'accès mémoires
  - Traitements temporaires

⇒ Permet de comprendre le coût d'évaluation du plan

Exemple de statistiques

Id	Opération	Name
0	SELECT STATEMENT	
1 *	NESTED LOOPS	
2	TABLE ACCESS FULL	Cours
3	TABLE ACCESS FULL	Auditeurs

(1) : filter(CODE = CODE_UE)

Statistics
0 recursive calls
100 db block gets
2000 consistent gets
300 physical reads

Différentes Statistiques

- Accès mémoire :
  - **consistent gets** : Nb d'accès à une donnée consistente en RAM (non modifiée)
  - **db block gets** : Nb d'accès à une donnée en RAM
- Accès physiques :
  - **physical reads** : Nb total de pages lues sur le disque
- **recursive calls** : Nb d'appels à un sous-plan (requête imbriquée)
- **redo size** : Taille du fichier de log produit (update)
- **sorts (disk)** : Nb d'opérations de tri avec au moins une écriture
- **sorts (memory)** : Nb d'opérations de tri en mémoire

$\Rightarrow$ Taux d'efficacité du cache : $1 - \frac{physical\ \ reads}{consistent\ \ gets\ +\ db\ \ block\ \ gets}$

Choix du meilleur plan d'exécution

- L'optimiseur a un PEL
- Choix des meilleurs opérateurs

$\Rightarrow$ Besoin de statistiques

Choix du meilleur opérateur

- Accès aux données : Vérification des index
  - Un ou plusieurs indexes en entrée
  - Partitions
- Jointures : Structure des données en entrée
  - Tables avec indexes
  - Tables partitionnées
  - Données triées
  - Ordre de la jointure : taille

Jeu de données

- **Table** *Cours*
  - 5000 n-uplets
  - 60 pages
  - Index BTree sur Responsable (Hauteur : 3, Ordre : 56)
  - Index BTree sur CODE (Hauteur : 2, Ordre : 150, code sur 10o)
  - 500 professeurs
- **Table** *Auditeurs*
  - 100 000 n-uplets
  - 300 pages
  - Index BTree sur CODE_UE (Hauteur : 3, Ordre : 150, $\phi_{code} = 10000$)
  - Index Bitmap sur Année (2 pages par vecteur)
  - 10 000 auditeurs
  - 10 années différentes (Histogramme)
- **Page** : 8 ko, $|\mathcal{M}| = 11$

Optimisation d'une requête

```
SELECT COUNT(*)
FROM Cours C, Auditeurs A
WHERE C.CODE_UE = A.CODE
    AND Responsable = 'Nicolas Travers'
    AND Annee = 2017;
```

Plan 1 : NESTED LOOPS

Id	Opération	Name
0	SELECT STATEMENT	
1 *	NESTED LOOPS	
2 *	TABLE ACCESS FULL	Cours
3 *	TABLE ACCESS FULL	Auditeurs

(1) : access(CODE = CODE_UE)
(2) : access(Responsable = 'Nicolas Travers')
(3) : access(Annee = 2017)

- Formule : $|Cours| + \left\lceil \frac{|Cours'|}{|\mathcal{M}|-2} \right\rceil \times |Auditeurs|$
- Accès séquentiel, $||Cours'|| = \frac{5000}{500} = 10$, $|Cours'| = 1$
- Formule : $60 + \left\lceil \frac{1}{11-2} \right\rceil \times 300 = 360$ I/O

Plan 2 : NESTED LOOPS inversé

Id	Opération	Name
0	SELECT STATEMENT	
1 *	NESTED LOOPS	
2 *	TABLE ACCESS FULL	Auditeurs
3 *	TABLE ACCESS FULL	Cours

(1) : access(CODE = CODE_UE)
(2) : access(Annee = 2017)
(3) : access(Responsable = 'Nicolas Travers')

- Formule : $|Auditeurs| + \left\lceil \frac{|Auditeurs'|}{|\mathcal{M}|-2} \right\rceil \times |Cours|$
- Accès séquentiel, $||Auditeurs'|| = \frac{100000}{10} = 10000$, Attribut CODE = 10o, $|Auditeurs'| = \left\lceil \frac{10000}{\left\lfloor \frac{8000}{3+(1+10)} \right\rfloor} \right\rceil = 18$
- Formule : $300 + \left\lceil \frac{18}{11-2} \right\rceil \times 60 = 420$ I/O

Remarques

- Accès aux données à partir des feuilles du plan
- Projection des données pour la jointure
(calcul de pages sans PCTFREE)

Plan 3 : NESTED LOOPS + INDEX RANGE SCAN

Id	Opération	Name
0	SELECT STATEMENT	
1 *	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	Cours
3 *	INDEX RANGE SCAN	Cours_Responsable_IDX
4 *	TABLE ACCESS FULL	Auditeurs

(1) : access(CODE = CODE_UE)
(3) : access(Responsable = 'Nicolas Travers')
(4) : access(Annee = 2017)

- Formule : $|\mathcal{I}_{resp}| + \Delta + \phi \times Sel_{resp} + \left\lceil \frac{|Cours'|}{|\mathcal{M}|-2} \right\rceil \times |Auditeurs|$
- Accès via index, $|\mathcal{I}| = 3, ordre : 56, Sel = \frac{1}{500}, \phi = ||Cours||$
- Formule : $3 + (\left\lfloor \frac{10}{56} \right\rfloor) + \frac{5000}{500} + \left\lceil \frac{1}{11-2} \right\rceil \times 300 = 313$ I/O

Utilisation des indexes

Id	Opération	Name
0	SELECT STATEMENT	
1 *	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	Auditeurs
3	BITMAP CONVERSION TO ROWIDS	
4 *	BITMAP INDEX SINGLE VALUE	Auditeurs_Annee
5 *	TABLE ACCESS FULL	Cours

(1) : access(CODE = CODE_UE)
(4) : access(Annee = 2017)
(5) : access(Responsable = 'Nicolas Travers')

Coût d'exécution

Id	Opération	Name
0	SELECT STATEMENT	
1 *	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	Auditeurs
3	BITMAP CONVERSION TO ROWIDS	
4 *	BITMAP INDEX SINGLE VALUE	Auditeurs_Annee
5 *	TABLE ACCESS FULL	Cours

- Lecture du bitmap : 2 I/O et 10 000 numéros de n-uplets
- Conversion en ROWIDs (lecture séquentielle de ROWIDs : 14) :

Remarques

- Calcul de sélectivité des indexes :
  - Sans statistiques : `Uniforme`
  - `Histogramme` (fréquence ou distribution)
  - Raffinage : `Dynamic Sampling`
- Quel est le coût si on utilise l'index Bitmap ?
- Modifier les méthodes d'accès si possible (avec calcul)
- Attention, un index peut empêcher d'utiliser un autre index en jointure

Plan 4 : NESTED LOOPS + INDEX RANGE SCAN

Id	Opération	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	Cours
3 *	INDEX RANGE SCAN	Cours_Responsable_IDX
4 *	TABLE ACCESS BY INDEX ROWID	Auditeurs
5 *	INDEX RANGE SCAN	Auditeurs_CODE_UE

(3) : access(Responsable = 'Nicolas Travers')
(4) : access(Annee = 2017)
(5) : access(CODE = CODE_UE)

- Formule : $|\mathcal{I}_{resp}| + \Delta + \phi \times Sel_{resp} + ||Cours'|| \times (|\mathcal{I}_{CODE}| + \Delta + \phi \times Sel_{CODE})$
- Accès via index + Jointure par index, $|\mathcal{I}_{resp}| = 3, Sel = \frac{1}{500}, \phi_{resp} = ||Cours||$, $|\mathcal{I}_{code}| = 3, ordre = 149, Sel = \frac{1}{5000}, \phi_{code} = 10000$
- Formule : $3 + (\lfloor \frac{10}{56} \rfloor) + \frac{5000}{500} + 10 \times (3 + (\lfloor \frac{100000 \times \frac{1}{5000}}{149} \rfloor) + \frac{10000}{5000})$
  $= 13 + 10 \times (3 + 0 + 2) = 63$ I/O

NESTED LOOPS + INDEX RANGE SCAN : Remarques

- Pour chaque cours, il y a 20 ROWID, mais le clustering factor prevoit que ces cours soit groupés sur 2 pages
- Quel est le PEP si la jointure est dirigée par Auditeurs ? Quel est son coût ?

Plan 5 : SORT MERGE JOIN

Id	Opération	Name
0	SELECT STATEMENT	
1*	MERGE JOIN	
2	SORT JOIN	
3	TABLE ACCESS BY INDEX ROWID	Cours
4 *	INDEX RANGE SCAN	Cours_Responsable_IDX
5	SORT JOIN	
6 *	TABLE ACCESS FULL	Auditeurs

(1) : access(CODE = CODE_UE)
(4) : access(Responsable = 'Nicolas Travers')
(6) : access(Annee = 2017)

- Formule : $|\mathcal{I}_{resp}| + \Delta + \phi \times Sel_{resp} + |Cours'| + 2|Cours'| \times \lceil \log_{|\mathcal{M}|-1}(|Cours'|) \rceil + |Auditeurs| + |Auditeurs'| + 2|Auditeurs'| \times \lceil \log_{|\mathcal{M}|-1}(|Auditeurs'|) \rceil + |Cours'| + |Auditeurs'|$
- Accès via index, $|Cours'| = 1$ ($|Cours'| \leq \mathcal{M}$), $|Auditeurs'| = 18$
- Formule : $3 + (\lfloor \frac{10}{56} \rfloor) + \frac{5000}{500} + 0 + 300 + 18 + 2 \times 18 \times \lceil \log_{10}(18) \rceil + 1 + 18 = 13 + 390 + 19 = 422$ I/O

SORT MERGE JOIN : Remarques

- Coût de lecture d'Auditeurs trop grand
- Tri peu couteux car en mémoire (filtrages et projections avant)
- Quel est le coût si $\mathcal{M} = 21$?

Plan 6 : HASH JOIN

Id	Opération	Name
0	SELECT STATEMENT	
1 *	HASH JOIN	
2	TABLE ACCESS BY INDEX ROWID	Cours
3 *	INDEX RANGE SCAN	Cours_Responsable_IDX
4 *	TABLE ACCESS FULL	Auditeurs

(1) : access(CODE = CODE_UE)
(3) : access(Responsable = 'Nicolas Travers')
(4) : access(Annee = 2017)

- Formule : $|\mathcal{I}_{resp}| + \Delta + \phi \times Sel_{resp} + |Auditeurs|$
- $|Cours'| \leq |\mathcal{M}| \Rightarrow MHJ$
- Formule : $3 + (\lfloor \frac{10}{56} \rfloor) + \frac{5000}{500} + 300 = 313$ I/O

Composition d'index

- Utilisation de deux indexes
  - Deux listes de ROWIDs
  - `AND-EQUAL` entre les listes
  - Accès aux données résultantes
- Quel index peut être utilisé ? Bitmap sur les années

PLAN 7 : NESTED LOOPS + INDEX + AND-EQUAL

Id	Opération	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	TABLE ACCESS BY INDEX ROWID	Cours
3 *	INDEX RANGE SCAN	Cours_Responsable_IDX
4	TABLE ACCESS BY INDEX ROWID	Auditeurs
5	AND-EQUAL	
6 *	INDEX RANGE SCAN	Auditeurs_CODE_UE
7	BITMAP CONVERSION TO ROWIDS	
8 *	BITMAP INDEX SINGLE VALUE	Auditeurs_Annee

(3) : access(Responsable = 'Nicolas Travers')
(6) : access(CODE = CODE_UE)
(8) : access(Annee = 2017)

- Formule : $|\mathcal{I}_{resp}| + \Delta + \phi \times Sel_{resp} + |Vecteur_{auditeurs}| + Conversion + ||Cours'|| \times (|\mathcal{I}_{CODE}| + \Delta + \phi \times Sel_{CODE} \times Sel_{annee})$
- Accès via index, $|\mathcal{I}| = 3, ordre : 56, Sel_{CODE} = \frac{1}{500}, Sel_{annee} = \frac{1}{10}, \phi = ||Cours||$, $|Vecteur_{Auditeurs}| = 2$, $Conversion \sim 3$
- Formule : $3 + (\lfloor \frac{10}{56} \rfloor) + \frac{5000}{500} + 2 + 3 + 10 \times (3 + 0 + \frac{10000}{5000 \times 10})$
  $= 13 + 5 + 10 \times (3 + 0 + 0,2) = 50$ I/O

AND-EQUAL : Remarques

- Intersection de liste empêche l'accès aux données inutiles
- Nécessaire avec beaucoup de données et plusieurs restrictions
- A-ton encore besoin d'accéder aux données Auditeurs ?
  SELECT COUNT(*)... Donc il suffit de compter le nombre de ROWIDs, sans accès aux données

PLAN 8 : - TABLES ACCESS BY INDEX ROWID

Id	Opération	Name
0	SELECT STATEMENT	
1	SORT AGGREGATE	
2	NESTED LOOPS	
3	TABLE ACCESS BY INDEX ROWID	Cours
4 *	INDEX RANGE SCAN	Cours_Responsable_IDX
5	AND-EQUAL	
6 *	INDEX RANGE SCAN	Auditeurs_CODE_UE
7	BITMAP CONVERSION TO ROWIDS	
8 *	BITMAP INDEX SINGLE VALUE	Auditeurs_Annee

(3) : access(Responsable = 'Nicolas Travers')
(6) : access(CODE = CODE_UE)
(8) : access(Annee = 2017)

- Formule : $|\mathcal{I}_{resp}| + \Delta + \phi \times Sel_{resp} + |Vecteur_{auditeurs}| + Conversion + ||Cours'|| \times (|\mathcal{I}_{CODE}| + \Delta) + Tri$
- Accès via index, $|\mathcal{I}| = 3$, $ordre : 56$, $Sel_{CODE} = \frac{1}{500}$, $Sel_{annee} = \frac{1}{10}$, $\phi = ||Cours||$, $|Vecteur_{Auditeurs}| = 2$, $Conversion \sim 3$, Sorti $< |\mathcal{M}| \Rightarrow Tri = 0$
- Formule : $3 + (\lfloor \frac{10}{56} \rfloor) + \frac{5000}{500} + 2 + 3 + 10 \times (3 + 0) + 0$
  $= 13 + 5 + 10 \times (3 + 0) + 0 = 48$ I/O

Conclusion sur l'optimiseur

- Vérification des méthodes d'accès
- L'accès aux données n'est pas toujours requis
- Choix de l'algorithme de jointure
- Combinaison des indexes
- Dépendant des statistiques sur les données

$\Rightarrow$ Nécessaire de bien choisir :

- Indexes (clé primaire, couvrant, combinaison)
- Taille/Type des attributs (Stockage, Projection, Index)
- Organiser physiquement les données (Non-dense, hachage, tri/clustering factor)

Pipeline

- Besoin de minimiser les écritures temporaires
  - Chaque opérateur à une sortie
  - Éviter l'écriture du résultat
- Besoin de minimiser la Mémoire Centrale
  - Affecter la mémoire sur tout un plan
  - À chaque opérateur nécessite une partie de $\mathcal{M}$
- Prenons deux opérateurs o_1 et o_2 en séquence
  - Que faire du résultat de o_1 ?

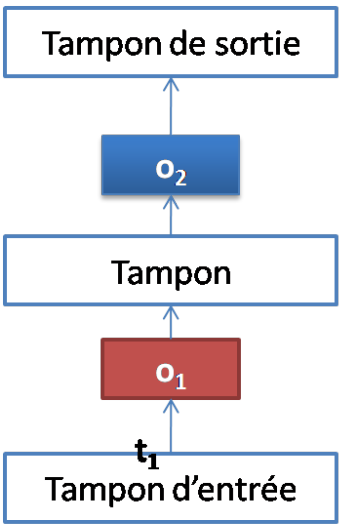
  $\Rightarrow$ Stocker sur le disque puis relire pour o_2
  Peut couter cher si le résultat grand

  $\Rightarrow$ Pipeliner o_1 avec o_2
  Résultat de o_1 directement passé dans o_2
- Pipeline impossible sur opérateurs *bloquants*.
  Quels sont les opérateurs bloquants ?
  SortMerge (+Join), Dédoublonnage, GroupBy, Agrégats

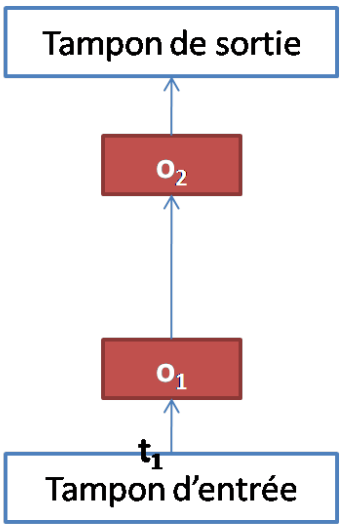
Principe

- Durant l'évaluation de o_1
  - A chaque n-uplet de **sortie** de o_1
  - Donné directement en **entrée** de o_2
  - *Production de o_1 consommé par o_2*
- Création d'une grande séquence d'opérateur en même temps (le plus possible)

Illustration

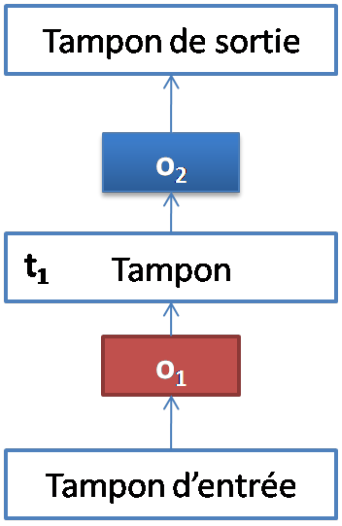


Sans pipeline

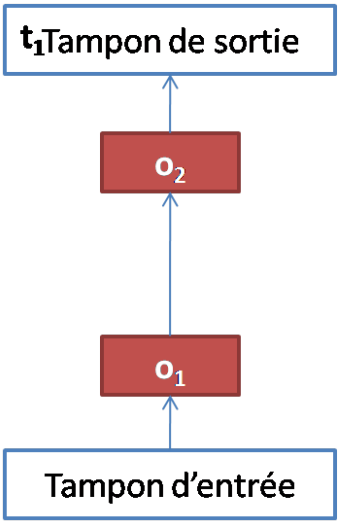


Avec pipeline

Illustration

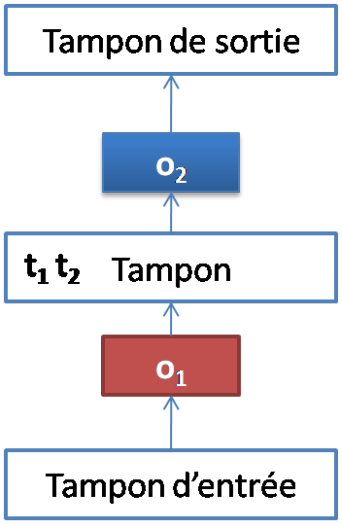


Sans pipeline

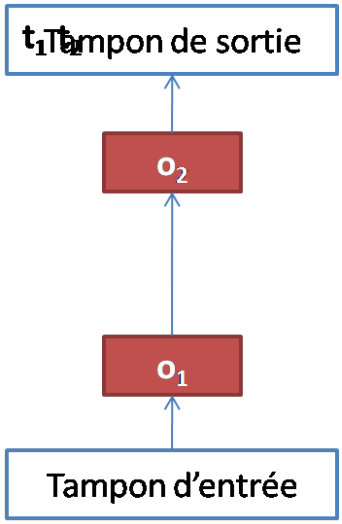


Avec pipeline

Illustration

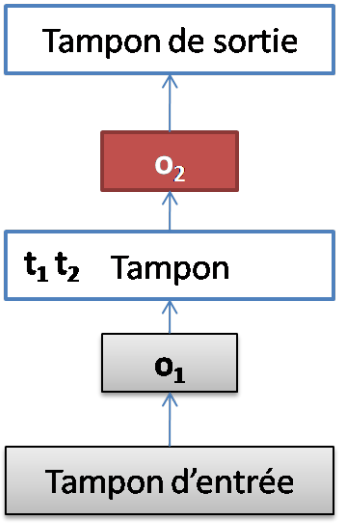


Sans pipeline

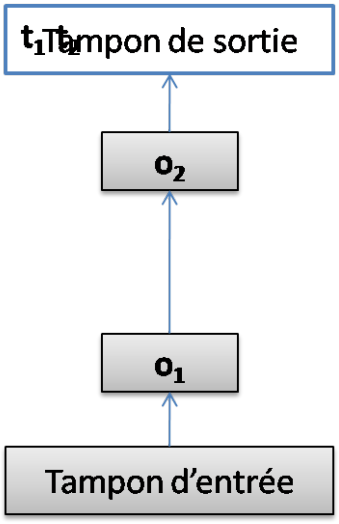


Avec pipeline

Illustration



Sans pipeline



Avec pipeline

Avantages

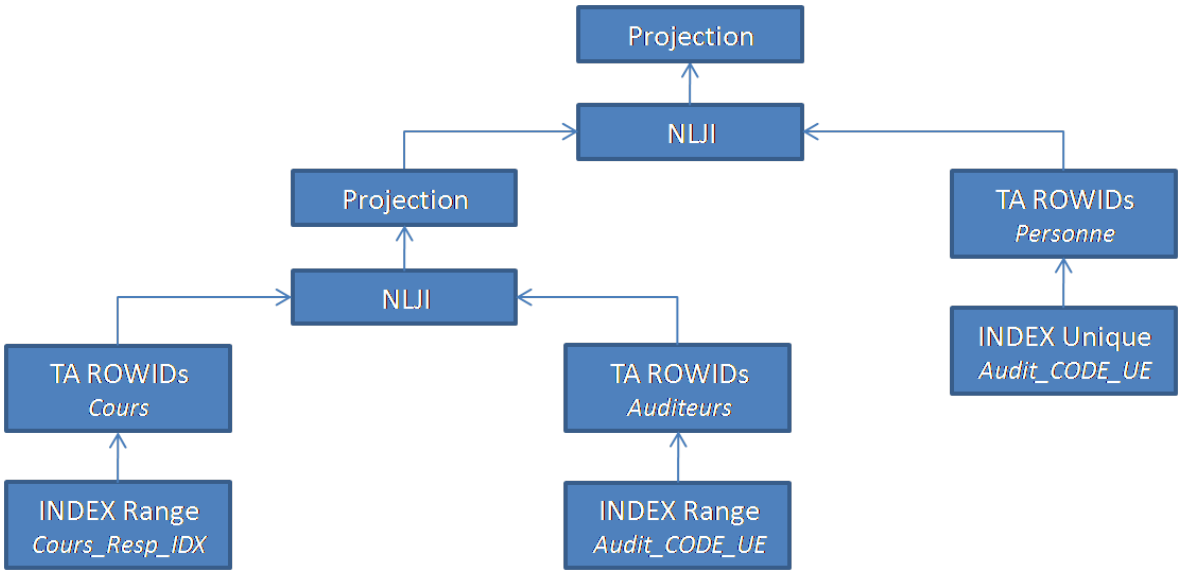
- Évite les lectures / écritures temporaires
- Optimise le traitement d'un tuple
- Allocation mémoire
  - Unique pour une séquence
  - Libérée plus rapidement

Exemple : double boucle imbriquée

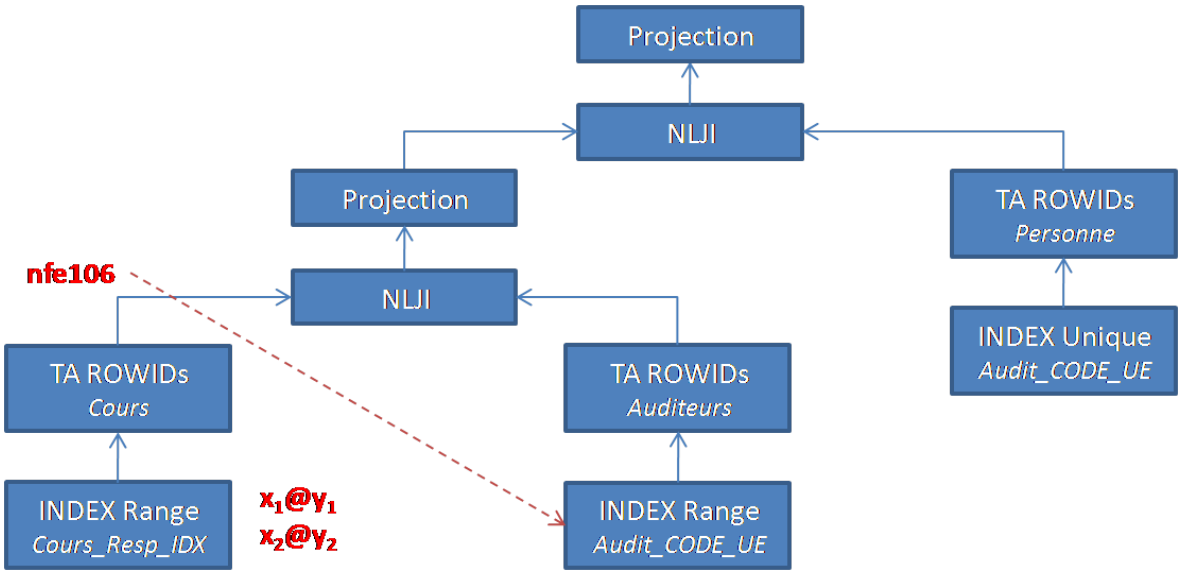
Id	Opération	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	NESTED LOOPS	
3	TABLE ACCESS BY INDEX ROWID	Cours
4 *	INDEX RANGE SCAN	Cours_Responsable_IDX
5 *	TABLE ACCESS BY INDEX ROWID	Auditeurs
6 *	INDEX RANGE SCAN	Auditeurs_CODE_UE
7	TABLE ACCESS BY INDEX ROWID	Personne
8 *	INDEX UNIQUE SCAN	Personne_ID

(4) : access(Responsable = 'Nicolas Travers')
(5) : access(Annee = 2010)
(6) : access(CODE = CODE_UE)
(8) : access(CODE_AUDITEUR = ID

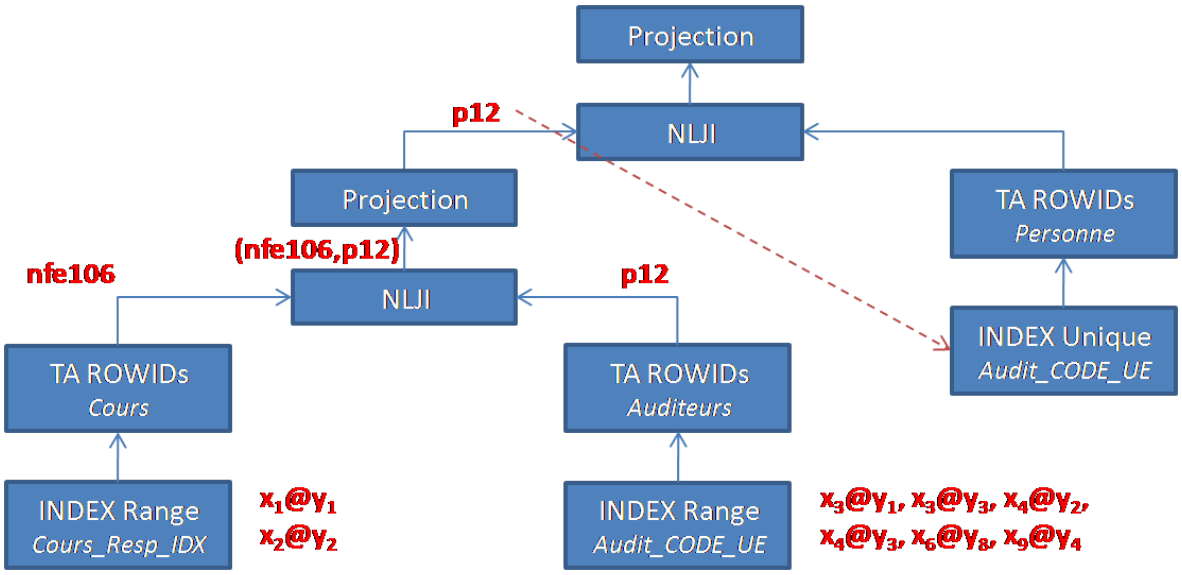
Illustration



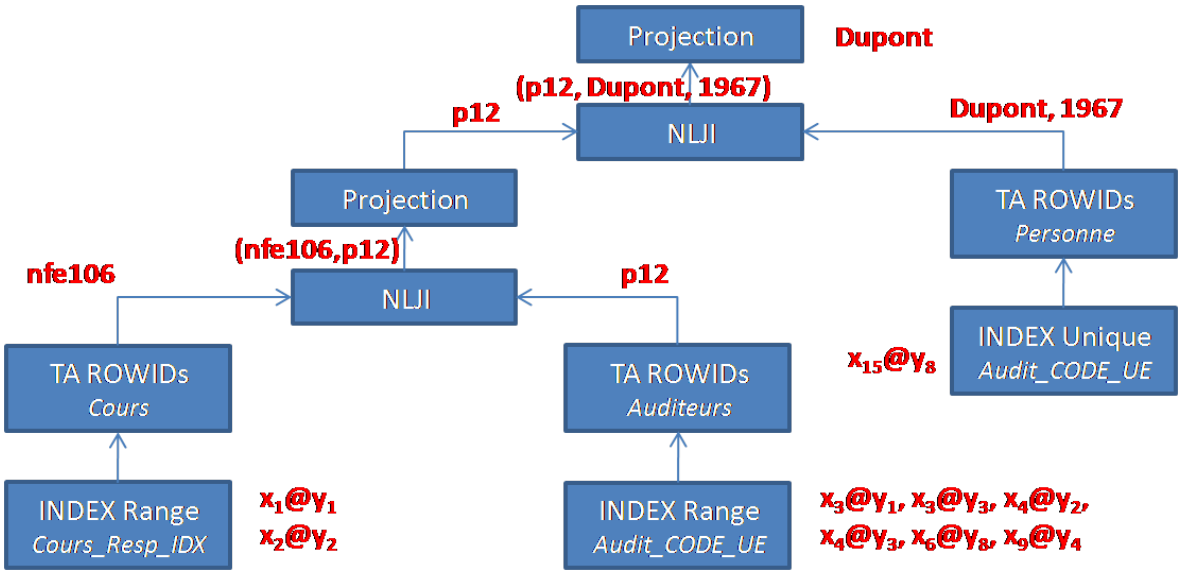
Illustration



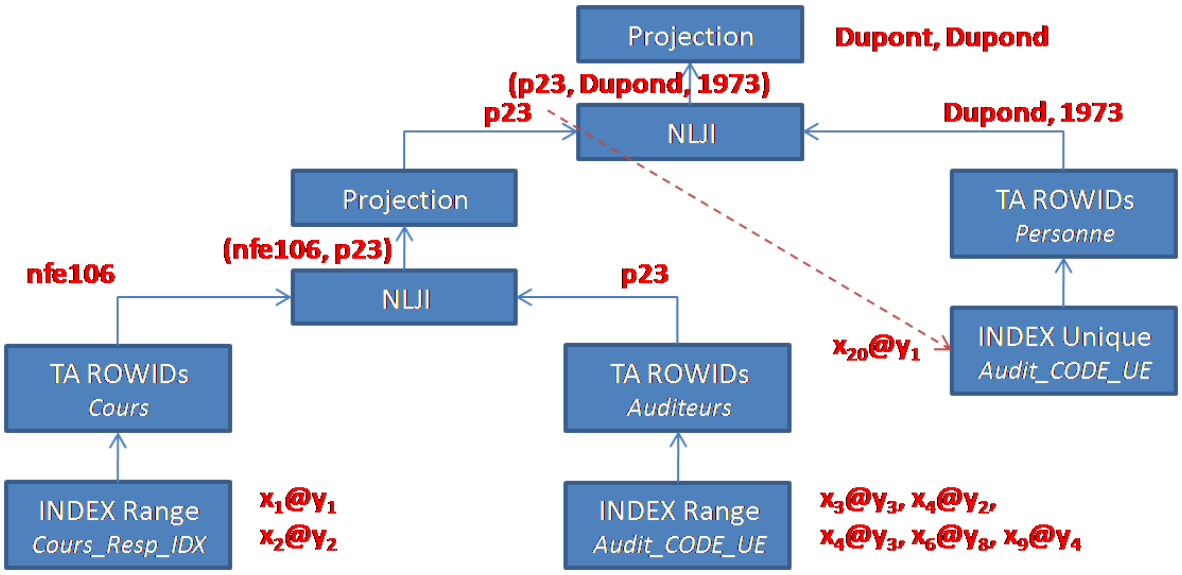
Illustration



Illustration



Illustration



Itérateurs

- À chaque opérateur est affecté un **Itérateur**
- Permet de produire / consommer à la demande
- Possibilité de créer des opérateurs 'Table' en pipeline : `Pipe-lined Functions`
- Besoins :
  - Type de n-uplet retourné
  - Type table contenant ces n-uplets
  - Fonction f retournant cette table en pipeline
  - Requête SQL avec `TABLE(`f`())` dans le FROM

Création de types

```
DROP TYPE pipelined_table;
DROP TYPE pipelined_row;

CREATE TYPE pipelined_row AS OBJECT (
    cours CHAR(5),
    NB_auditeur NUMBER(3,0)
);
/
CREATE TYPE pipelined_table IS TABLE OF pipelined_row;/
```


Création de la fonction en pipeline

```
CREATE OR REPLACE FUNCTION pipelined_function
        (pcours IN CHAR(5), pannee IN NUMBER)
        RETURN pipelined_table PIPELINED AS
    CURSOR c (ccours CHAR(5), cannee NUMBER) AS
        SELECT COUNT(*) AS NB
            FROM Auditeur WHERE cours = ccours AND annee = cannee;
BEGIN
    FOR i IN c(pcours, pannee) LOOP
        PIPE ROW(pipelined_row(pcours, i.NB));
    END LOOP;
    RETURN;
END;
/
```

Utilisation de la fonction en pipeline

```
SELECT *
FROM TABLE(pipelined_function('NFE106', 2017));
```


Conclusion

Pipeline

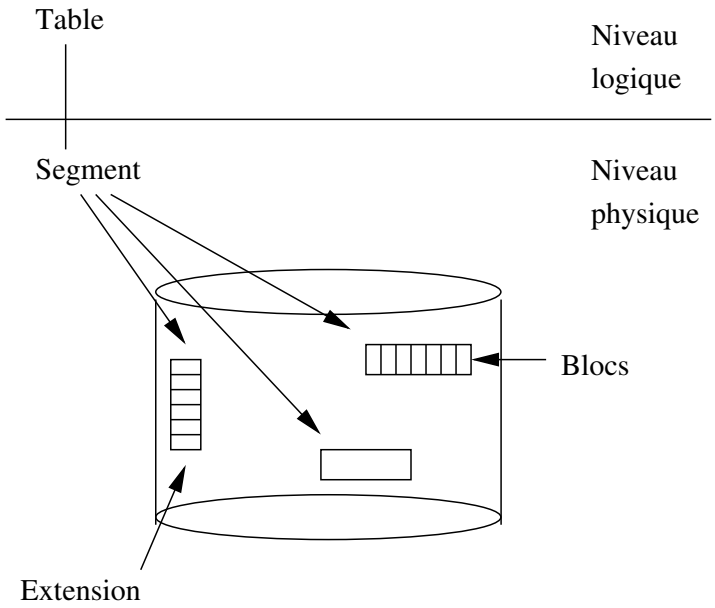
1. Permet d'optimiser le résultat du premier tuple
2. Réduit le nombre d'écriture intermédiaire
3. Possibilité de créer des résultats localement optimisés (*Pipe-lined Functions*)
4. Problème : Allocation de plus de ressources pour une seule requête ($|M|$ est réparti sur l'ensemble des opérateurs du plan)

Représentation physique des données

Structures physiques dans ORACLE :

1. **Bloc** : Unité physique d'E/S (entre 1KO et 8KO).
   - Multiple de la taille des blocs du système d'exploitation
2. **Extension** : Ensemble de blocs *contigus*
   - Contient un même type d'information
3. **Segment** : ensemble d'*extensions*
   - Stockent un objet logique (une table, un index ...)

Tables, segments, extensions et blocs



Le Segment ORACLE

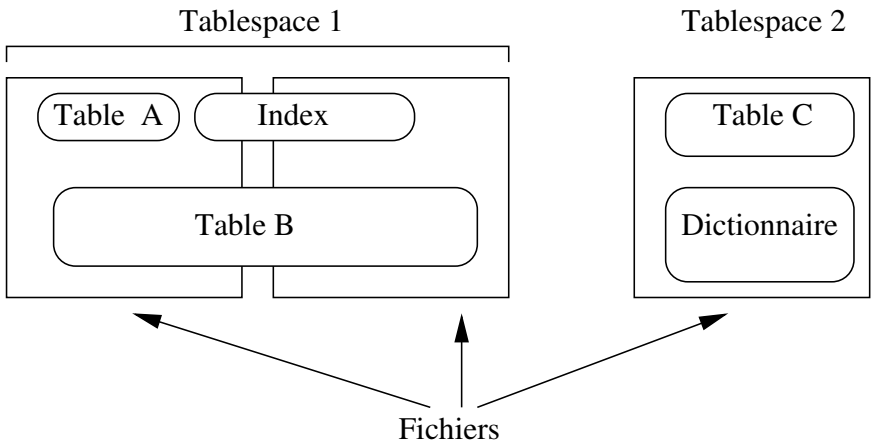
- Zone physique contenant un objet logique
- Quatre types de segments :
  1. Segment de données (Table, *Cluster*, Partition)
  2. Segment d'Index
  3. *Rollback Segment* utilisé pour les transactions
  4. Segment temporaire (utilisé pour les tris, résultats temporaires)

Base ORACLE, fichiers et *TABLESPACE*

1. **Base de données physique** : Ensemble de fichiers
2. **Base de données logique** : Ensemble de `TableSpace`
   - Divisés par l'administrateur
   - *tablespace* un ou plusieurs fichiers (physiques)

   Permet :
   1. Contrôle l'emplacement physique des données (Dictionnaire sur un disque, Données utilisateur sur un autre)
   2. Facilite la gestion (sauvegarde, protection, etc)

Exemple de TableSpace



Stockage des données

Deux stockages possibles :

1. **Indépendant :** Segments alloués automatiquement à la table. Paramètres :
   1. Sa taille initiale
   2. Le pourcentage d'espace libre dans chaque bloc
   3. La taille des extensions
2. En **cluster** : Regroupement de données

TABLESPACE : définition

```
CREATE TABLESPACE tabspace2
    DATAFILE 'diskC:tabspacefile2.dat' SIZE 2G
    DEFAULT STORAGE (INITIAL 80K NEXT 160K
              MINEXTENTS 1 MAXEXTENTS 999
              PCTINCREASE 20)
```

CREATE TABLE

```
CREATE TABLE Cours (
    CODE_UE NUMBER PRIMARY KEY
              USING INDEX TABLESPACE usersa,
    Responsable VARCHAR2(50),
    NB_AUDITEURS NUMBER )
TABLESPACE tabspace_2
STORAGE (INITIAL 160K PCTINCREASE 50)
PCTFREE 10 PCTUSED 75
```


Les *clusters*

- Un cluster est un regroupement de données sur valeurs
- *Avantage* : Optimiser les jointures
  - Ex : Cours $\bowtie_{UE=CODE\_UE}$ Auditeurs
  - Groupement sur le CODE_UE
  - Stockage des données (n-uplets) dans un segment cluster
  - Création d'index sur CODE_UE
- *Désavantage* : Augmente la taille de la table (peu de clés par page)
  $\Rightarrow$ Augmente les accès en parcours séquentiel.

Exemple de cluster

Page	CODE_UE	Intitule	Responsable	NB_auditeurs
P1	NFE106	Ingénierie de Base de Données	Nicolas Travers	20
		CODE_AUDITEUR	**Annee**	
		A1	2017	
		A6	2017	
		A15	2010	
		A23	2009	

Page	CODE_UE	Intitule	Responsable	NB_auditeurs
P2	NFP107	Initiation à la Base de Données	Michel Crucianu	100
		CODE_AUDITEUR	**Annee**	
		A1	2009	
		A6	2009	
		A3	2007	
		A12	2017	
		A23	2008	

CLUSTER avec B+Tree

- `CREATE CLUSTER cluster_UE`
  `(CODE_UE VARCHAR2(6))`
  `SIZE 2K`
  `INDEX`
  `STORAGE (INITIAL 100K NEXT 50K)` [26]
- `CREATE INDEX ind_UE ON CLUSTER cluster_UE` [27]

26. *SIZE* est la taille de toutes les données pour une clé donnée.

CREATE TABLE

- ```
  CREATE TABLE Cours (
        CODE VARCHAR2(6) PRIMARY KEY,
        intitule VARCHAR(32),
        Responsable VARCHAR(64))
  CLUSTER cluster_UE(CODE);
  ```
- ```
  CREATE TABLE Auditeurs (
        CODE_UE VARCHAR2(6),
        CODE_AUDITEUR VARCHAR2(10),
        annee NUMBER(4), note NUMBER(2),
        PRIMARY KEY (CODE_AUDITEUR, CODE_UE))
  CLUSTER cluster_UE(CODE_UE);
  ```

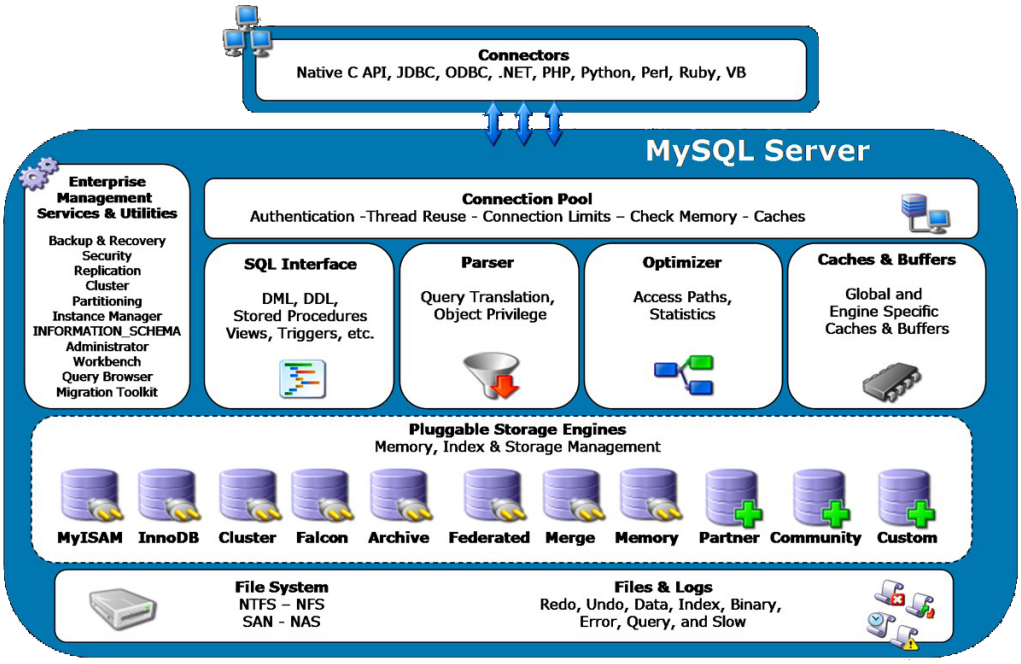

CLUSTER avec Hachage

- ```
  CREATE CLUSTER hash_UE
        (CODE_UE VARCHAR2(6))
        HASH IS CODE_UE
        HASHKEYS 1000 SIZE 2K ;
  ```
- *HASH IS* (optionnel) : Clé à hacher
- *HASHKEYS* : Nombre de valeurs à hacher
  ```

MySQL - Historique

- 1994 - Michael (Monty) Widenius & David Axmark
- 1995 - First release
- 2001 - MySQL 3.3 - InnoDB - moteur transactionnel
- 2002 - MySQL 4 - unions, prepare statements, requêtes imbriquées
- 2005 - MySQL 5 - trigger, vues, curseurs, transactions, partitions, réplication des n-uplets
  SUN + MySQL
- 2009 - MySQL 5.4 - Net amélioration des performances
- 2010 - Oracle achète SUN (et donc MySQL), version 5.5
- Utilisé par Google, Facebook, Amazon, elections américaines

MySQL Server : Une architecture *Pluggable*



Source : MySQL Performances 2009

MySQL : Moteurs de Stockage

- CREATE TABLE nomTable (...) ENGINE `nomMoteur`
- Moteurs de stockages principaux :
  - **MyISAM** (par défaut - Sun/`MySQL`)
  - **InnoDB** (1° transactionnel - `Oracle`)
  - `ARCHIVE` : Archivage de données (compression lors de l'insertion)
  - `MEMORY` : Moteur de stockage entièrement en mémoire
  - `NDB` : BD en réseau (grappe de serveurs)
  - `Maria` : MyISAM + propriétés ACID

MyISAM

- Premier moteur de MySQL
- Moteur de stockage par défaut
- Avantages :
  - Peu volumineux et rapide
  - Requêtes de type *COUNT*
  - Index textuel : FULLTEXT
- Désavantages :
  - Pas de concurrences d'accès
  - Reprise sur panne difficile

MyISAM : Stockage

- Stockage sous formes de tas
- Pas de cache de données
- Modifications directement visibles
- Composé de trois fichiers :
  1. *table*.**frm** : Description
  2. *table*.**MYD** : Données
  3. *table*.**MYI** : Indexes (dont celui sur la clé primaire)
- Indexes sont gardés dans le cache

MyISAM : Paramètres

- A la fin du *CREATE TABLE*, rajouter :
  - DATA DIRECTORY = '/disk1/myData'
  - INDEX DIRECTORY = '/disk2/index/myIndex' ;
- Pourquoi mettre sur deux disques physiques index et données ?

MyISAM : *Full Text*

- Index efficace en recherche textuelle [28]
  - Texte indexé sous forme de BTree
  - `LIKE` : Recherche exacte
  - `MATCH/AGAINST` : Recherche avec tri des résultats par pertinence

    Exemple : SELECT *, MATCH (Intitule) AGAINST ('Base de Données') AS Score FROM Cours ORDER BY score DESC ;

    Possibilité en mode 'BOOLEAN' : plus efficace

28. Etudié en NFE204

MyISAM : Pannes et Concurrence

- Pas de moteur transactionnel
  - Un seul processus d'écriture
- Garanties de *Recovering*, mais lent
  - Journal de sauvegarde incrémentale
- Indexes en cache
  - Plus performants
  - Problèmes d'incohérence

InnoDB

- Moteur par défaut à partir de MySQL 5.5 (version efficace)
- Index clé primaire efficace
- Clés étrangères
- Transactionnel
- Multi-threadé

InnoDB : Stockage

- Composé de deux fichiers :
  1. *table*.**frm** : Description
  2. *table*.**ibd** : Données (+ index secondaire + undo logs)
- Indexes sont gardés dans le cache

InnoDB : BTree

- Index sur la clé primaire : Non-dense
  (besoin de clé de petite taille)
- Stockage des données dans les **feuilles** de l'arbre
- Cache sur les noeuds de l'index
- Autres index gardent la valeur de la clé primaire
  (et non le ROWID)

MySQL : Choix du moteur

MyISAM **Beaucoup lectures / Recherche textuelle**

InnoDB **Read+Write / Transactions / Accès clé primaire**

NDB **Petites Transactions**, traitements parallèles

MEMORY **Uniquement en mémoire**

SQL Server - Historique

- 1989 - Sybase et Microsoft - sous Unix et OS/2 (IBM)
- 1994 - Microsoft garde SQL Server - Windows NT
- 2008 - Version `Katmaï`, intègre toutes les librairies Windows

Stockage

- Plusieurs pages par partition ;
- Extensions de 8 pages ;
- Plusieurs partitions par table/objet ;
- Une table correspondant à un ensemble de fichiers :
  - `.mdf` - primary data ;
  - `.ndf` - secondary data (text, multimedia...) ;
  - `.ldf` - log data ;
- Une base de données est un ensemble de fichiers.

Extensions et SQL Server

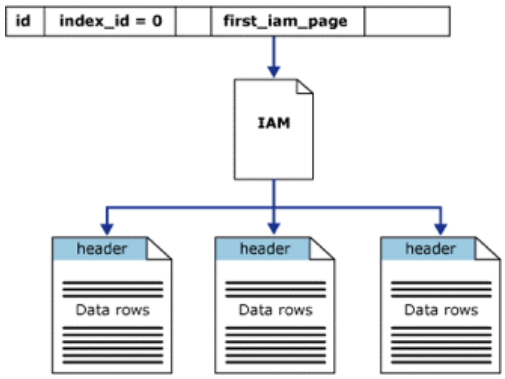
- 8 pages contigües (non paramétrable) ;
- 2 types :
  - Uniforme - un seul objet ;
  - Mixte - jusqu'à 8 objets - un objet par page ;
- Optimisation du traitement des adresses :
  - Lecture contigües jusqu'à 8 extensions (512 Ko)
  - Anticipe les demandes des requêtes

IAM

- `IAM` - Index Allocation Map
- Une séquence pour chaque objet
- Désignés par le 'File Header' (1/fichier)
- Détaille la localisation des pages
  (Données ou Indexes)

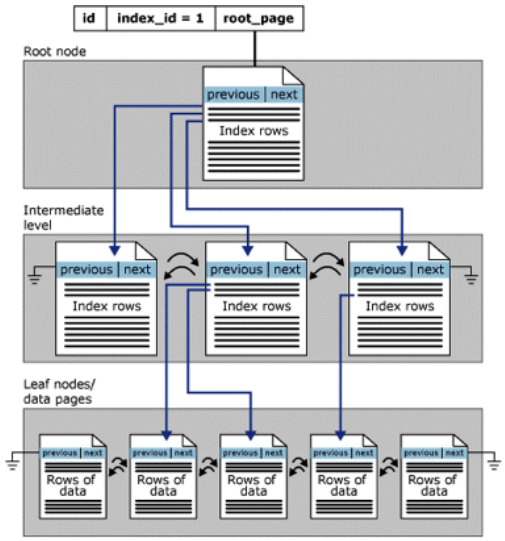
Heap Tables

- Tables **sans** index non-dense ;
- Pas d'ordre (tuples ou pages) ;
- Page IAM pour retrouver les extensions ;



Clustered Tables

- Tables **avec** BTree non-dense
  - Sur clé unique ou primaire ;
  - Peut être couvrant ;
- Chainage à tous les niveaux ;
- Créé automatiquement pour les Vues indexées ;



Clustered Tables : Exemple

```
CREATE TABLE Enseignant (
     ID int NOT NULL PRIMARY KEY,
     Nom nvarchar(100) NOT NULL UNIQUE CLUSTERED,
     Prenom nvarchar(100) NOT NULL,
     datenaiss date NOT NULL
);
```


Expressions régulières

```
SELECT zip FROM zipcode
WHERE regexp_like (zip, '[^[:digit:]]');
```

DB2 - Historique

- 1982 - IBM sort SQL/DS (DB2 mainframe, System R)
- 1992 - DB2 sur OS2
- 1995 - DB2 2.1 sur Unix, Windows NT, OS2
- 2006 - DB2 9, *Deep Compression*, Intégration de XML

Compression

- ↗ compression = ↘ pages = ↘ d'Entrées/Sorties
- Gain de place par compression :
  1. *Tuples* : Algorithme `Lempel-Ziv` ;
  2. *Indexes* : Idem ;
  3. *Tables temporaires* : Idem ;
  4. *Backups* : Idem ;
  5. *Valeurs* : NULL et chaines vides ;
  6. *XML* : librairie de balises (index) ;

Compression de tuples

- Algorithme `Lempel-Ziv` modifié (LZ) :
- Basé sur un dictionnaire :
  - Symboles (12 bits) ;
  - Un par table ($\approx$ 75 Ko) ;
  - Efficace pour de fortes répétitions de valeurs ;
- Augmente le coût CPU ;

Lempel-Ziv (LZ) - Exemple

Données : (Taille totale : 206o)

CODE	Intitule	Responsable	NB	Taille
NFE106	Ingénierie de Base de Données	Nicolas Travers	20	**52o**
NFP107	Initiation à la Base de Données	Michel Crucianu	100	**55o**
NFE204	Base de Données Avancées 1	Michel Scholl	30	**49o**
NFE205	Base de Données Avancées 2	Michel Crucianu	20	**50o**

Dictionnaire :

Chaine	Code
NFE	01
10	02
6	03
Ingénierie	04
de	05
Base de Données	06
Nicolas Travers	07
20	08
NFP	09
7	10

Chaine	Code
Initiation	11
à la	12
Michel	13
Crucianu	14
100	15
4	16
Avancées	17
1	18
Scholl	19
30	20
2	21

Compression :

01	02	03	04	05	06	07	08	
09	02	10	11	12	06	13	14	15
01	08	16	06	17	18	13	19	20
01	08	16	06	17	21	13	14	08

Taille totale :
420 bits ≈ 53o

Comment compresser ?

1. `COMPRESS` doit être mis pour sur la table
   `CREATE TABLE Enseignant (…) COMPRESS YES;`
   `ALTER TABLE Enseignant … COMPRESS YES;`
2. Un dictionnaire de compression doit exister
   `REORG Enseignant KEEPDICTIONARY;`
   `REORG Enseignant RESETDICTIONARY;`
3. Estimation de la place economisable :
   `INSPECT ROWCOMPESTIMATE TABLE NAME Enseignant ;`
   Donne le nombre de pages economisées.

Compression résultats

- **Espace :** 50 à 75% de pages sauvées en moyenne ;
- **Performance pour systèmes orientés :**
  - CPU : 20-30% de gain ;
  - I/O : 10% de perte ;

DB2 : suppléments

Possibilité d'ajouter :

- `PCTFREE` (comme `Oracle`) ;
- Index `CLUSTER` (non-dense) ;

PostgreSQL - Historique

- 1985 - Le SGBD Ingres de *Michael Stonebraker (Berkeley)* devient Postgres
- 1995 - Intégration de SQL : PostgreSQL
- 2005 - Fonctionne sous Windows
- 2012 - Version 9.1.4

Caractéristiques

- Création de types de données (Héritages de types)
- Création de modules/opérateurs supplémentaires (évolutif)
- Programmation étendues sur le moteur avec PL/pgSQL
- Traitement des données externe possible (programmes compilés)
  - DBLink : Connexion sur plusieurs instances PostgreSQL
  - PostGIS : Module de données spatiales

SQLite - Historique

- 2000 - Conçu par *Richard Hipp* en contrat avec la `United States Navy` (BD légère pour lancement de missiles)
- 2001 - v2.0, intégration de BTree et transactions
- 2005 - v3.0, EXPLAIN, + compatible SQL, contraintes
- 2012 - v3.7, *In Memory Database*
- Utilisé par Adobe, Airbus, Mac OSX, Dropbox, Firefox, Google, Skype, plateformes smartphone...

Caractéristiques

- Base de données légère (350 ko)
  - Sans installation du SGBD (librairie à appeler)
  - Sans administration complexe
- Fonctionne sur tous les systèmes
- Pas de serveur / client (ligne de commande / librairie)
- Typage dynamique (pas au niveau des colonnes, mais des valeurs)
- Impossible de supprimer des colonnes

Recommandations

- Taille des pages :
  - 1024 pour linux (par défaut)
  - 4096 pour Windows NTFS
  - Avant la première table en ligne de commande : `PRAGMA page_size=4096;`
- Interface client : `SQLite Manager`
- Lors de la création de la base, ne pas utiliser l'extension .sdb (executable Windows). Préférer `.db`

Indexes

	BTree (dense)	**Cluster** (non-dense)	**Hash**	**Bitmap**	**Full-Text**	**RTree** (2D)
Oracle	oui	IOT	Cluster Hash	oui	oui	oui
MySQL	oui	Memory InnoDB (PK)	NDB	non	MyISAM	MyISAM
SQL Server	oui	PK (par defaut)	non	non	oui	non
DB2	oui	Cluster	Cluster Hash	IDB	oui	non
PostgreSQL	oui	Cluster	oui	oui	oui	PostGIS
SQLite	oui	non	non	non	oui	oui

Qu'est-ce qu'une mauvaise requête ?

- Ne tient pas compte des données
  (Répartition, attributs inutiles, répétitions)
- Ne tient pas compte des indexes
  (fonctions, statistiques, calculs complexes)
- Ne tient pas compte de l'organisation physique
  (tri, cluster, partition)
- Génère trop de résultats temporaires
  (requêtes imbriquées corrélées)
- Trop déclarative
  (précalculs nécessaires)

Attributs inutiles

SELECT * FROM Auditeur ;
⇒ SELECT CODE_UE FROM Auditeur ;

Oubli du prédicat de Jointure

SELECT COUNT(*) FROM Cours C, Auditeur A
WHERE Intitule='NFE106' AND ANNEE=2017 ;

⇒ SELECT COUNT(*) FROM Cours C, Auditeur A
WHERE Intitule='NFE106' AND ANNEE=2017
AND C.CODE=A.CODE_UE ;

Plan d'exécution avec un produit cartésien.

Mauvaise utilisation de l'index couvrant

Index couvrant sur `Cours.(Intitule, Responsable)`.
SELECT Intitule FROM Cours
WHERE Responsable='Nicolas Travers' ;
⇒ Index couvrant sur `Cours.(Responsable, Intitule)`.

Précalcul : Requête imbriquée corrélée

```
SELECT intitule FROM Cours c
WHERE annee =
        (SELECT MAX(annee)
        FROM Auditeur a WHERE
        c.CODE = a.CODE_UE) ;
```

⇒
```
SELECT intitule FROM Cours c,
        (SELECT CODE_UE, MAX(annee) AS max
        FROM Auditeur
        GROUP BY CODE_UE) a
WHERE c.CODE = a.CODE_UE and a.max=c.annee ;
```

La requête imbriquée est exécutée pour chaque tuple *Cours*

Précalcul : Jointure groupée

```
SELECT CODE_UE, intitule, count(*)
FROM Cours c, Auditeur a
WHERE c.CODE = a.CODE_UE
GROUP BY CODE_UE, intitule ;
```

⇒
```
SELECT CODE_UE, intitule, NB FROM Cours c,
        (SELECT CODE_UE, COUNT(*) AS NB
                FROM Auditeur
                GROUP BY CODE_UE) a
WHERE c.CODE = a.CODE_UE ;
```

Utilisation de Fonctions sur index

Lieu (X, Y, NOM), Btree sur X et sur Y

```
SELECT NOM FROM lieu
WHERE SQRT(POW (x - 400, 2) + POW (y - 400, 2))<= 100 ;
```

Id	Opération	Name
0	SELECT STATEMENT	
1*	TABLE ACCESS FULL	lieu

Lieu (X, Y, NOM), Btree sur X et sur Y

```
⇒ SELECT NOM FROM lieu
WHERE (x BETWEEN 300 AND 500) AND
      (y BETWEEN 300 AND 500) AND
      (POW (x - 400, 2) + POW (y - 400, 2))
          <= 10000 ;
```

Id	Opération	Name
0	SELECT STATEMENT	
1	TABLE ACCESS BY ROWIDS	lieu
2*	AND-EQUAL	
3*	INDEX RANGE SCAN	BTREE_X
4*	INDEX RANGE SCAN	BTREE_Y

Fonction Date

```
SELECT order_created FROM orders
WHERE TO_DAYS(CURRENT_DATE())
                - TO_DAYS(order_created) <= 7 ;

⇒ SELECT order_created FROM orders
WHERE order_created >=
        CURRENT_DATE() - INTERVAL 7 DAY ;
```


Index sur valeurs numériques

```
SELECT COUNT(*) FROM Auditeur Where annee = '2017' ;

⇒ SELECT COUNT(*) FROM Auditeur Where annee = 2017 ;
```

DISTINCT = Tri

SELECT **CODE** FROM Cours ;

$$\neq$$

SELECT **DISTINCT CODE_UE** FROM Auditeur ;

$\Rightarrow$ SELECT **CODE** FROM Cours
WHERE **CODE IN (SELECT CODE_UE FROM Auditeur)** ;

Tuple aléatoire

SELECT CODE, intitule FROM Cours
ORDER BY RAND() LIMIT 1 ;

$\Rightarrow$ SELECT count(CODE) FROM Cours INTO @m ;
SET @z := FLOOR(RAND()*@m)+1 ;
SELECT CODE, intitule FROM Cours
LIMIT @z, 1 ;

Problème

- Fonction sur attribut => pas d'index ;

⇒ Indexation du résultat de la fonction ;
- Evite le parcours séquentiel des données ;
- Evite des calculs complexes (PL/SQL) sur un ou plusieurs attributs ;
- Peut changer de nombreux plans d'exécutions ;
- Peut contenir des expressions arithmétiques ;
- Si le résultat est matérialisé, l'optimiseur peut estimer la sélectivité de la fonction ;

Exemple

- `SELECT * FROM Cours WHERE nb_groupe(NbAuditeur) = 3 ;`
- `CREATE FUNCTION nb_groupe (NB_eleve INT) RETURN INT IS BEGIN RETURN CEIL(NB_eleve / 40); END;/`
- `CREATE INDEX nb_groupe_cours ON Cours(nb_groupe(NbAuditeur)) ;`

La vue relationnelle

- Table virtuelle calculée à partir d'une requête SQL
- ```
  CREATE VIEW vue_AvgEleve AS
      SELECT CODE, AVG(NbEleve) as AvgNbEleve
      From Cours
      GROUP BY CODE ;
  ```
- Apparaît comme une table :
  - `vue_AvgEleve(CODE, AvgNbEleve)`
- Permet de :
  - Cacher des données sensibles
  - Simplification des accès (précalcul des définitions de tables)
  - Montrer des données statistiques
  - Indépendance logique du schéma d'origine
- Modifications rarement possibles (monotable + clé primaire)
- ⚠ Optimisé indépendamment du reste de la requête

Vues matéralisées

- Le résultat de la vue (requête) est stockées physiquement
- La matérialisation est interrogée à la place de la requête
- Problème de fraicheur de la vue (périodique / sur demande)
- Utilité :
  - Pré-agrégation de données (optimisation)
  - Pré-récupération des données (distribuées)
  - Sécurisation des données
- ⇒ Traitement optimisé du résultat
- ⚠ Mise à jour couteuse
  ```

Vues matérialisée : syntaxe

- CREATE **MATERIALIZED** VIEW vue_compte_eleve
  <propriétés de stockage> (TABLESPACE / ORGANIZATION / CLUSTER)
  BUILD (IMMEDIATE | DEFERRED) (immédiat / sur REFRESH)
  [<mise à jour>] (REFRESH date / PK / ROWID / ROLLBACK SEGMENT)
  AS SELECT CODE, ANNEE, COUNT(*) AS NB_ELEVE
  FROM Cours JOIN Auditeurs ON CODE = CODE_UE
  GROUP BY CODE, ANNEE ;

  Syntaxe complète :
  http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_6002.htmici

Comment Inspirer/Suggérer l'optimiseur ?

- Le `HINT` force l'utilisation d'un opérateur
- A n'utiliser qu'après analyse du plan EXPLAIN
- Se place après la clause SELECT
- Expression :
  - SELECT /*+ hint */
  - SELECT /*+ hint(argument) */
  - SELECT /*+ hint(argument-1 argument-2) */

Exemple

SELECT **/*+ *index* (a BTREE_code_UE) */** a.CODE_AUD
FROM Auditeur a
WHERE CODE_UE = 'NFE106' ;

Types de HINT

1. Heuristique de l'optimiseur
2. Accès aux données
3. Ordre de jointure
4. Algorithme de jointure
5. Exécution en parallèle
6. Star Transformation
7. Vues
8. Divers

Modifier l'heuristique de l'optimiseur

- **ALL_ROWS**
  Invoque l'optimiseur produisant le plan le plus optimisé (batch ou data warehouse).
- **FIRST_ROWS(n)**
  Invoque l'optimiseur produisant un plan qui optimise les *n* premiers résultats.
- **CHOOSE / RULE**
  Choix du plan d'exécution avec/sans statistiques *(Deprecated depuis 9g)*

Accès aux données

- **CLUSTER**
  Accès par l'index de cluster
- **FULL**
  Parcours séquentiel (SelSeq)
- **HASH**
  Hachage pour une table partitionnée
- **ROWID**
  Identification de tuples par ROWIDs
- **INDEX / NO_INDEX**
  Force l'index sur une table
- **INDEX_ASC / INDEX_DESC**
  Feuilles de l'index ASC/DESC
- **INDEX_COMBINE**
  Combinaison de bitmaps
- **INDEX_FFS**
  FAST FULL SCAN
- **AND_EQUAL**
  Fusion de liste de ROWIDs
- **INDEX_JOIN**
  Jointure de liste de ROWIDs

Algorithmes de jointure

- **LEADING**
  Jointure des tables dans l'ordre donné
- **ORDERED**
  Jointure des tables dans l'ordre de déclaration
- **USE_NL / USE_NL_WITH_INDEX / USE_HASH / USE_MERGE + NO_...**
  Force l'utilisation de la jointure sélectionnée sur la table donnée
- **NL_AJ / HASH_AJ / MERGE_AJ**
  NOT IN : anti-jointure
- **NL_SJ / HASH_SJ / MERGE_SJ**
  EXISTS : semi-jointure
- **NO_PUSH_SUBQ / PUSH_SUBQ**
  La sous-requête est évaluée le plus tôt/tard possible

Parallélisation

- **PARALLEL / NOPARALLEL**
  Force le pipeline entre deux tables
- **PARALLEL_INDEX / NOPARALLEL_INDEX**
  Parallèlisation du Fast Full Scan

Star Transformation

- **STAR_TRANSFORMATION** [29]
  Réécriture de requêtes en sous-requêtes optimisées (Conjonction de prédicats, fusion de vues, désimbrication de requêtes, réécriture de vues matérialisées, ...)
- **FACT / NO_FACT**
  La table ciblée est considée comme la table de *fait* pour la "`Star Transformation`"
- **USE_CONCAT / NO_EXPAND**
  Force l'utilisation de l'opération UNION-ALL pour des prédicats OR
- **UNNEST/NO_UNNEST**
  Désimbrication des sous-requêtes

29. Schéma en étoile

Vues

- **REWRITE / NOREWRITE**
  Force l'utilisation d'une vue matérialisée. (NOREWRITE : A utiliser avec concurrence d'accès et sans vues matérialisées)
- **MATERIALIZE**
  Force la matérialisation de la table temporaire
- **PUSH_PRED / NO_PUSH_PRED**
  Force le passage d'un prédicat de jointure dans la vue

Divers

- **CARDINALITY / SELECTIVITY**
  Donne la cardinalité/sélectivité de la table ciblée
- **QB_NAME**
  Donne un nom à un opérateur qui peut être réutilisé par la suite
- **APPEND / NOAPPEND**
  `direct-path INSERT` (fin de table)/ `serial INSERT` (liste de places libres)
- **CACHE**
  Place la table ciblée dans le cache
- **RESULT_CACHE**
  Place le résultat de la requête dans le cache pour une requête future
- **DYNAMIC_SAMPLING**
  Niveau de précision (0 à 10) de l'optimiseur (sélectivité des prédicats).
- **DRIVING_SITE**
  Exécution de la requête dans un site différent de celui proposé par l'optimiseur

Notations

- $|\mathcal{R}|$: Nb de pages de la relation $\mathcal{R}$
- $|\mathcal{R}'|$: Nb de pages de la relation $\mathcal{R}$ après modification d'un opérateur
- $||\mathcal{R}||$: Nb de n-uplets de la relation $\mathcal{R}$
- $||\mathcal{R}'||$: Nb de n-uplets de la relation $\mathcal{R}$ après sélection
- $|\mathcal{I}|$: Nb de pages à lire pour une tranversée d'index (ie. hauteur de l'arbre)
- *ordre* : Ordre du l'arbre B $\mathcal{I}$
- ϕ : *clustering factor*
- *Sel* : Sélectivité d'un attribut
- Δ : Nb de feuilles de $\mathcal{I}$ sélectionnées
- $|\mathcal{M}|$: Nb de pages tampons en MC pour la lecture

Rappels de formules (1/2)

- $|Tuple| : 3 + x \times (|X| + 1) + y \times (|Y| + 2) \ldots$
- $|DataBlock| : (8192 - entete) \times (\frac{100 - PCTFREE}{100})$
- $|Table| : \left\lceil \frac{||tuples||}{\lfloor \frac{|DataBlock|}{|tuple|} \rfloor} \right\rceil$
- Ordre d'un BTree : $\left\lfloor \frac{|DataBlock|}{3+x+|cle|+10} \right\rfloor \div 2$
- $|\mathcal{I}| : \lceil \log_{ordre}(||tuple||) \rceil$
- INDEX RANGE SCAN : $|\mathcal{I}| + \Delta + \phi \times Sel$
  - Index dense : $\phi \simeq ||S||$
  - Index non-dense : $\phi \simeq |S|$
  - $\Delta = \left\lfloor \frac{||S|| \times Sel}{ordre} \right\rfloor$

Rappels de formules (2/2)

- $Tri_{\mathcal{R}}$: Accès à $\mathcal{R} + |\mathcal{R}'|$
  $+2 \times |\mathcal{R}'| \times \left\lceil \log_{|\mathcal{M}|-1}(|\mathcal{R}'|) \right\rceil$ (si $|\mathcal{R}'| \geq \mathcal{M}$)
- `NESTED LOOPS` : Accès à $\mathcal{R} + \left\lceil \frac{|\mathcal{R}'|}{|\mathcal{M}|-2} \right\rceil \times |\mathcal{S}|$
- `NESTED LOOPS + INDEX RANGE SCAN` :
  Accès à $\mathcal{R} + ||\mathcal{R}'|| \times (|\mathcal{I}| + \phi \times Sel + \Delta)$
- 2 x SORT JOIN + MERGE JOIN :
  $Tri_{\mathcal{R}} + Tri_{\mathcal{S}} + |\mathcal{R}'| + |\mathcal{S}'|$
- HASH JOIN (memory : $\mathcal{R}' < \mathcal{M}$) :
  Accès à $\mathcal{R}$ + Accès à $\mathcal{S}$
- HASH JOIN (grace : $\mathcal{M} < \mathcal{R}' < \mathcal{M}^2$) :
  Accès à $\mathcal{R}$ + Accès à $\mathcal{S} + 2 \times |\mathcal{R}'| + 2 \times |\mathcal{S}'|$